29th International Conference on DNA Computing and Molecular Programming

DNA 29, September 11-15, 2023, Tohoku University, Sendai, Japan

Edited by

Ho-Lin Chen

Constantine G. Evans

Editors

Ho-Lin Chen
National Taiwan University, Taipei, Taiwan
holinchen@ntu.edu.tw

Constantine G. Evans
Maynooth University, Ireland
cevans@costinet.org

ACM Classification 2012
Theory of computation → Models of computation; Applied computing → Molecular structural biology; Applied computing → Biological networks; Information systems → Information storage systems

ISBN 978-3-95977-297-6

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-297-6.

Publication date
September, 2023

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.DNA.29.0

ISBN 978-3-95977-297-6 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Papers

29th International Conference on DNA Computing and Molecular Programming (DNA 29).
Editors: Ho-Lin Chen and Constantine G. Evans
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at DNA 29: the 29th International Conference on DNA Computing and Molecular Programming. The conference was held during September 11–15, 2023, at Tohoku University, Sendai, Japan, and was organized under the auspices of the International Society for Nanoscale Science, Computation, and Engineering (ISNSCE). The DNA conference series aims to draw together researchers from the fields of mathematics, computer science, physics, chemistry, biology, and nanotechnology to address the analysis, design, and synthesis of information-based molecular systems.

Papers and presentations were sought in all areas that relate to biomolecular computing, including, but not restricted to: algorithms and models for computation on biomolecular systems; computational processes in vitro and in vivo; molecular switches, gates, devices, and circuits; molecular folding and self-assembly of nanostructures; analysis and theoretical models of laboratory techniques; molecular motors and molecular robotics; information storage; studies of fault-tolerance and error correction; software tools for analysis, simulation,and design; synthetic biology and in vitro evolution; and applications in engineering, physics, chemistry, biology, and medicine.

Authors who wished to orally present their work were asked to select one of two submission tracks: Track A (full paper) or Track B (one-page abstract with supplementary document). Track B is primarily for authors submitting experimental or theoretical results who plan to submit to a journal rather than publish in the conference proceedings. We received 61 submissions for oral presentations: 25 submissions to Track A and 36 submissions to Track B. Each submission was reviewed by at least two reviewers, with most reviewed by three or more. The Program Committee accepted 10 papers for Track A (40%) and 11 papers for Track B (31%). We also received 108 submissions for Track C (poster), of which six were selected as additional oral presentations by the Program Committee. This volume contains the papers accepted for Track A.

We express our sincere appreciation to our invited speakers: Petra Berenbrink, Chunhai Fan, Masami Hagiya, Olgica Milenkovic, Yusuke Sato, and Georg Seelig. We thank all of the authors who contributed papers to these proceedings, and those who presented papers and posters during the conference. Last, but by no means least, the editors are especially grateful to the members of the Program Committee and the additional invited reviewers for their hard work in reviewing the papers on a tight deadline and for providing insightful and constructive comments to the authors.

Ho-Lin Chen
Constantine Evans
September 2023

29th International Conference on DNA Computing and Molecular Programming (DNA 29).
Editors: Ho-Lin Chen and Constantine G. Evans
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Steering Committee

Anne Condon (chair)	University of British Columbia
Masami Hagiya	University of Tokyo
Natasha Jonoska	University of Southern Florida
Matthew Lakin	The University of New Mexico
Satoshi Murata	Tohoku University
John H. Reif	Duke University
Grzegorz Rozenberg	University of Leiden
Rebecca Schulman	Johns Hopkins University
Friedrich Simmel	Technical University Munich
David Soloveichik	The University of Texas at Austin
Andrew Turberfield	Oxford University
Shelley Wickham	The University of Sydney
Erik Winfree	California Institute of Technology
Damien Woods	Maynooth University
Hao Yan	Arizona State University

29th International Conference on DNA Computing and Molecular Programming (DNA 29).
Editors: Ho-Lin Chen and Constantine G. Evans
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Ho-Lin Chen (co-chair)	National Taiwan University
Constantine Evans (co-chair)	Maynooth University
Jonathan Bath	University of Oxford
Luca Cardelli	University of Oxford
Yuan-Jyue, Chen	Microsoft
Anne Condon	The University of British Columbia
David Doty	University of California, Davis
Abeer Eshra	Maynooth University
Cody Geary	Aarhus University
Masami Hagiya	The University of Tokyo
Rizal Hariadi	Arizona State University
Hope Amber Johnson	University of British Columbia
Ibuki Kawamata	Tohoku University
Satoshi Kobayashi	The University of Electro-Communications
Akinori Kuzuya	Kansai University
James Lathrop	Iowa State University
Chenxiang Lin	Yale University
Dongsheng Liu	Tsinghua University
Satoshi Murata	Tohoku University
Tosan Omabegho	
Pekka Orponen	Aalto University
Thomas Ouldridge	Imperial College London
Matthew Patitz	University of Arkansas
Luca Piantanida	Boise State University, Idaho
Lulu Qian	California Institute of Technology
Trent Rogers	Maynooth University
Lorenzo Rovigatti	Sapienza University of Rome
Dominic Scalise	Washington State University
Nicholas Schabanel	CNRS - École Normale Supérieure de Lyon
Jo Schaeffer	Google
Robert Schweller	University of Texas Rio Grande Valley
Shinnosuke Seki	The University of Electro-Communications
Shalin Shah	Bloomberg
William Shih	Harvard University
Jaimie Stewart	University of California, Los Angeles
Petr Šulc	Arizona State University
Masahiro Takinoue	Tokyo Institute of Technology
Chris Thachuk	University of Washington
Grigory Tikhomirov	University of California, Berkeley
Boya Wang	California Institute of Technology
Shelley Wickham	University of Sydney
Sungwook Woo	Pohang University of Science and Technology

Additional Reviewers for Tracks A and B

Samantha Borje	University of Washington
Alex Dack	Imperial College London
Hannah Earley	University of Cambridge
Timothy Gomez	Massachusetts Institute of Technology
Matteo Guareschi	California Institute of Technology
Daniel Hader	University of Arkansas
Carina Imburgia	University of Washington
Tiernan Kennedy	University of Washington
Austin Luchsinger	University of Texas at Austin
Cadence Pearce	University of Washington
Chandler Petersen	University of Washington
Samson Petrosyan	University of California, Berkeley
Durham Smith	University of California, Berkeley
Anli Tang	University of California, Los Angeles
Tao Zhang	Yantai University
Olivia Zhou	California Institute of Technology

Organizing Committee for DNA 29

Satoshi Murata (chair)	Tohoku University
Taro Toyota	The University of Tokyo
Shin-ichiro M. Nomura	Tohoku University
Takashi Nakakuki	Kyushu Institute of Technology
Akinori Kuzuya	Kansai University
Ibuki Kawamata	Tohoku University
Shogo Hamada	Tohoku University
Masahiro Takinoue	Tokyo Institute of Technology
Takuya Mabuchi	Tohoku University
Keita Abe	Tohoku University

Sponsors

International Society for Nanoscale Science, Computation, and Engineering
Grants-in Aid for Transformative Research Areas (A), KAKENHI
SECOM Science and Technology Foundation
Sendai Tourism, Convention and International Association
The NOVARTIS Foundation (Japan) for the Promotion of Science
Nihon Techno Service Co., Ltd
Springer
Support Center for Advanced Telecommunications Technology Research
Nihon Gene Research Laboratories, Inc
Oxford Instruments / Andor

Minimum Free Energy, Partition Function and Kinetics Simulation Algorithms for a Multistranded Scaffolded DNA Computer

Ahmed Shalaby
Hamilton Institute, Department of Computer Science, Maynooth University, Ireland

Chris Thachuk
Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA, USA

Damien Woods
Hamilton Institute, Department of Computer Science, Maynooth University, Ireland

Abstract

Polynomial time dynamic programming algorithms play a crucial role in the design, analysis and engineering of nucleic acid systems including DNA computers and DNA/RNA nanostructures. However, in complex multistranded or pseudoknotted systems, computing the minimum free energy (MFE), and partition function of nucleic acid systems is NP-hard. Despite this, multistranded and/or pseudoknotted systems represent some of the most utilised and successful systems in the field. This leaves open the tempting possibility that many of the kinds of multistranded and/or pseudoknotted systems we wish to engineer actually fall into restricted classes, that do in fact have polynomial time algorithms, but we've just not found them yet.

Here, we give polynomial time algorithms for MFE and partition function calculation for a restricted kind of multistranded system called the 1D scaffolded DNA computer. This model of computation thermodynamically favours correct outputs over erroneous states, simulates finite state machines in 1D and Boolean circuits in 2D, and is amenable to DNA storage applications. In an effort to begin to ask the question of whether we can naturally compare the expressivity of nucleic acid systems based on the computational complexity of prediction of their preferred energetic states, we show our MFE problem is in logspace (the complexity class L), making it perhaps one of the simplest known, natural, nucleic acid MFE problems. Finally, we provide a stochastic kinetic simulator for the 1D scaffolded DNA computer and evaluate strategies for efficiently speeding up this thermodynamically favourable system in a constant-temperature kinetic regime.

2012 ACM Subject Classification Theory of computation → Models of computation; Applied computing → Physical sciences and engineering

Keywords and phrases thermodynamic computation, model of computation, molecular computing, minimum free energy, partition function, DNA computing, DNA self-assembly, DNA strand displacement, kinetics simulation

Digital Object Identifier 10.4230/LIPIcs.DNA.29.1

Supplementary Material *Software (Source code)*: `https://dna.hamilton.ie/software.html`

Funding Damien Woods and Ahmed Shalaby were supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 772766, Active-DNA project), and Science Foundation Ireland (SFI) under grant numbers 20/FFP-P/8843 and 18/ERCS/5746. Chris Thachuk was supported by a US NSF grant (CCF 2211792) and a Faculty Early Career Development Award from NSF (CCF 2143227).

Acknowledgements We thank Abeer Eshra for extensive discussions on the thermodynamics and kinetics of SDC-based strand displacement and for experimental advice, Tristan Stérin for thoughts on the SDC model, Dave Doty for helpful comments, and Constantine Evans, David Soloveichik and Erik Winfree for insightful algorithmic discussions.

1 Introduction

Efficient algorithms play a crucial role in the design, analysis and engineering of nucleic acid systems including DNA computers and DNA nanostructures. Some decades ago the beautiful relationship between pseudoknot-free nucleic acid secondary structures[1] and dynamic programming algorithms was established [28, 17, 33]. For just one or more DNA/RNA strands, there are asymptotically exponentially many (in number of bases) distinct secondary structures those strands may adopt. Despite this, for the case of a single DNA or RNA strand, dynamic programming can be used to efficiently, in polynomial time, predict important global properties of the system, such as its pseudoknot-free minimum free energy (MFE) or partition function. Intuitively, the MFE is the energy of the most favoured[2] structure(s) of the system, and the partition function is the sum of the Boltzmann-weighted energy of each secondary structure of the system.

Background. The free energy $\Delta G(S) = \sum_{l \in S} \Delta G(l) + (n-1)\Delta G^{\text{assoc}}$ of an n-strand secondary structure S is a sum of empirically-obtained [22] free energies $\Delta G(l)$ of a set of *loop* features of the secondary structure such as stack loops, bulge loops, and many others [7]. $\Delta G^{\text{assoc}} > 0$ is an entropic penalty for bringing strands together. The MFE of a set Ω of unpseduoknotted secondary structures is simply $\min_{S \in \Omega} \Delta G(S)$, and the partition function is a (typically large positive) number $Q = \sum_{S \in \Omega} e^{-\Delta G(S)/k_{\text{B}}T}$ where k_{B} is Boltzmann's constant in units of kcal/(mol $\cdot$ K) and T is temperature in Kelvin. Q is typically used as a normalisation factor to calculate the probability of any pseudoknot-free secondary structure S at equilibrium: $p(S) = (e^{-\Delta G(S)/k_{\text{B}}T})/Q$.

Dynamic-programming based MFE and partition function algorithms provide a firm basis for some sophisticated DNA sequence design tasks. For example, we might use partition function calculations in a design feedback loop with the goal of lowering the probability of unwanted ("off-target") secondary structures, and we may use MFE and/or partition function to improve the design of desired ("on-target") secondary structures [31, 10, 3, 7, 20].

As noted, there are fast MFE and partition function algorithms for systems consisting of a single strand. But for the case of multiple interacting strands, even unpseudoknotted, the situation is more nuanced: if we have a constant number (independent of input length) of strands, there is a polynomial time dynamic programming partition function algorithm [7], although, somewhat surprisingly, we don't know of one for MFE [4], since there are overcounting complications when there are multiple copies of the same strand. But if the system is multistranded in the most general sense, where the number of strands is given as part of the input (i.e. non-constant), unpseudoknotted MFE was recently shown to be both NP-hard and hard to approximate (APX-hard) and so is unlikely to have a polynomial time algorithm, even for a simple energy model (partition function is unknown for this case) [4]. Furthermore, if pseudoknots are permitted, MFE determination becomes NP-hard for a large class of reasonable energy models, even for a single strand [1, 15].

---

[1] We have an ordered list of DNA strands, where each strand is a sequence of DNA bases ordered in 5' to 3' direction, giving a total order on all DNA bases in the system. Then, a secondary structure is simply a set of pairs of the form (i, j) interpreted as "base i is paired with base j" (by convention $i < j$). We may permute the order of the strands, and a structure is called unpsedoknotted if there exists a strand order where such base pairs "do not cross", i.e. a strand ordering where there are no two pairs (i, j) and (i', j') such that $i' \in [i, \ldots, j]$ and $j' \notin [i, \ldots, j]$. This point is more intuitively explained using polymer graphs (see Figure 1).

[2] In physics and chemistry, more negative free energy generally means more favoured.

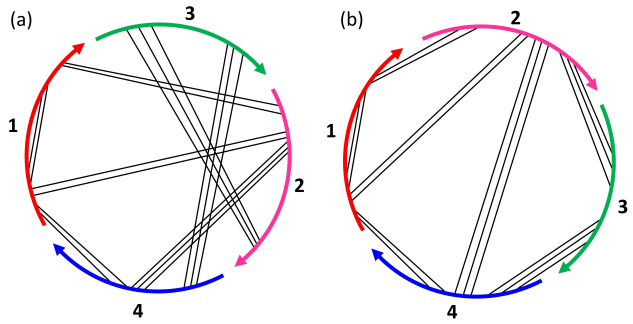


Figure 1 (a) Polymer graph for a secondary structure over the strand set $\{1, 2, 3, 4\}$ with strand ordering 1324. Some crossings are shown. (b) By simply reordering to 1234 we get another polymer graph for the same strand set and secondary structure, but without crossings. The existence of a strand ordering that yields a crossing-free diagram implies that this secondary structure is unpseudoknotted.

Despite the theoretical road-block of algorithmic hardness, multistranded and pseudoknotted self-assembly systems are some of the most successful DNA nanostructure and DNA computing paradigms: examples include DNA origami [21], RNA origami [12], and single-stranded tile systems [29, 31]. In these cases the design process has not been via full de novo algorithmic simulation of the thermodynamics, but by other methods including decomposition into smaller pieces more amenable to analysis [31], or by intuition-based whiteboard drawings, or some other method. Another key point is that such systems, and many others, are typically designed at a domain level of abstraction, where we ignore the details of DNA sequences and imagine a set of strands, each with one or more domains, and where several domains are imagined to have identical free energy, and domains that should not bind never in fact bind.

Motivation. But perhaps many engineered multistranded and/or pseudoknotted systems make use of design concepts, such as modularity, abstraction, principled thermodynamics, or some other intuitively-attractive principles that make their design and analysis algorithmically tractable? In other words, despite the above-cited algorithmic hardness results, we may ask if there are sub-classes of multistranded and/or pseudoknotted DNA systems, that on the one hand are interesting (experimentally implementable, make complex nanostructures, are capable of computation, etc.) yet on the other hand allow for efficient, e.g. polynomial time, MFE and partition function algorithms? Digging further into a computational nuance: since the ability to calculate MFE and/or partition function seems to require detailed (combinatorial/ enumerative) knowledge of a system one may wonder if there is a hierarchy of nucleic acid computing systems, classified according to difficulty of computing MFE and/or partition function, where higher levels corresponds to more expressive molecular computers capable of stronger computation than lower-level systems?

Scaffolded DNA Computer. Here we look at a specific multistranded system. Stérin, Eshra and Woods [24] introduced a thermodynamically favoured [8] model of computation called the Scaffolded DNA Computer (SDC). The model is computationally expressive and has theoretical thermodynamic benefits over most forms of molecular computing. In one dimension (1D SDC) the model is capable of simulating finite state automata and transducers, and in 2D simulates arbitrary Boolean circuits [5] (but proved for a slightly different model).

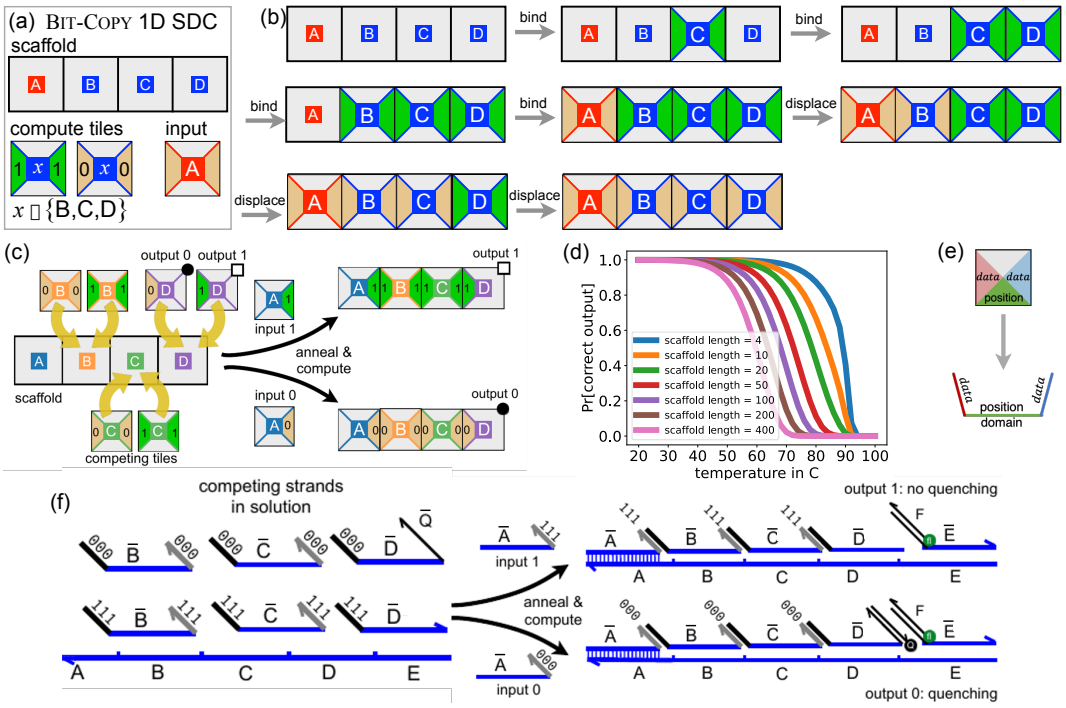


Figure 2 Thermodynamic model of the 1D scaffolded DNA computer (SDC) illustrated using a simple BIT-COPY program (a wire) of length $N = 4$ and maximum tiles per scaffold position $k = 2$. (a) Example BIT-COPY system for a length $N = 4$ scaffold. The system consists of 7 tiles: 2 per scaffold position B, C and D and 1 input tile at scaffold position A. (b) An example execution: starting from the empty scaffold, a tile may *bind* or *displace* an existing tile: in both cases the total number of matching bonds (scaffold-tile or tile-tile) should monotonically increase. 5 *bind* steps followed by 2 *displace* steps are shown. The final configuration has 4 scaffold-tile bonds and 3 tile-tile bonds, making it the most favourable. (c) Depending on whether input 0 or 1, the BIT-COPY system eventually reaches the most favoured (all-0 or all-1) configuration. (d) Using our partition function algorithm to simulate an anneal: simulations, for various scaffold lengths showing probability of target (correct output for BIT-COPY) with respect to temperature, using experimentally-relevant parameters, see Appendix E. (e) Schema for converting tiles to domain-level DNA strand abstraction. (f) Domain level abstraction of BIT-COPY system. 10 programs have been experimentally implemented in this model [24], the strand diagrams show some implementation features (3-bit domains for data and computing, as well as fluorescence/quencher reporting complexes).

Figure 2(a–c) describes the model at an abstract tile-level of abstraction suited to programming and theoretical analysis (see Section 2.1 for a domain-level model definition). This computational model borrows key principles from DNA origami [21] (low concentration scaffold, high concentration strands that bind to the scaffold), but breaks one origami principle by allowing non-scaffold strands to bind to each other, albeit somewhat weakly (no stronger than scaffold-tile binding). These principles yield a thermodynamically favoured model of molecular computation where the target (output) structure, is most favoured.

The SDC has been implemented experimentally [24] in 1D, showcasing a total of 10 programs that solve problems such as ADDITION of two 4-bit numbers, PARITY of an 8-bit input (is the number of 1s odd?), and GRAPH REACHABILITY (is there a path in an input graph from a source node s to a target node t). One future aim is to scale up to significantly larger 1D and 2D systems, by exploiting the thermodynamic favourability of the scaffolded design principle. However, to do that we need new algorithmic design tools since these systems are multistranded in 1D and 2D, and pseudoknotted in 2D.

Main results. We prove a number of results for the domain-based 1D SDC model. The following two theorems are respectively proven in Sections 3 and 4 (see Figure 2 for N, k).

▶ **Theorem 1.** There is an $O(k^2N)$ time algorithm to determine the domain-level MFE for a 1D SDC of length N with $\leq k$ computation strands competing at each scaffold domain.

▶ **Theorem 2.** There is an $O(k^2N)$ time algorithm for the domain-level partition function for a 1D SDC of length N with $\leq k$ computation strands competing at each scaffold domain.

Our third result opens a door to the research direction of classifying the complexity of (computational or not) nucleic acid systems based on the computational complexity of computing MFE/partition function for those systems (proof in Appendix D):

▶ **Theorem 3.** Given a 1D SDC $\mathcal{S}$ of length $N \in \mathbb{N}$, with $k = O(1)$ computing strands per scaffold domain and bounded domain energies ($\Delta G(d) \in O(1)$, for a domain d binding to its complement), and a finite-precision decimal r, the problem of deciding whether the domain-level MFE of $\mathcal{S}$ is $\leq r$ is in the complexity class L.

Our fourth contribution is exploratory: In Section 5 we give a stochastic kinetic simulator for 1D SDC systems. We evaluate the kinetics of the 1D SDC, and go on to test two strategies to speed up kinetics, the first using extra components and the second exploiting varying concentrations, both showing some speed-up. This initiates the kinetic study of the 1D SDC model, where one imagines operating the system at a fixed temperature allowing the system to move towards equilibrium via designed kinetic pathways. This stands in contrast to previous work [24] which focused on using the 1D SDC thermodynamically: heat it up, cool it down!

1.1 Related work

The SDC leverages thermodynamic favourability principles from DNA origami [21], and computational abilities from self-assembly [31] and strand displacement systems such as the scaffolded SIMD||DNA system [26, 27] as well as theory-based systems [8]. We wish to create computational tools to evaluate future strategies for both thermodynamic computation, and constant-temperature (kinetic) computation. But the SDC has a unique combination of self-assembly and strand displacement features that make direct use of existing tools problematic and led us to design bespoke thermodynamic (MFE and partition function) prediction algorithms, and a kinetic simulator, all built on the same energy model.

SDC self-assembly features include: growth of large target structures, a large combinatorically-described set of intermediate structures, hassle-free scaffold-mediated seeded growth with in-built avoidance of unwanted off-scaffold nucleation (computational domains bind weakly, hence enumeration/simulation of *all* complexes is not desired nor even relevant). SDC strand-displacement features include: use of toeholds, and displacement domains which are sometimes desired to work along standard toehold-mediated strand displacement style pathways, but other times not, *depending on our choice of operating mode* – so far experimental work used a simple, fast anneal [24], without needing careful kinetic control!

Related work on MFE and partition function was already discussed above. Multistranded MFE is NP-hard [4], a result that holds in a simple domain-level model. Our polynomial time MFE algorithm (Theorem 1) is also for a multistranded domain-level system; however, we exploit the 1D SDC binding structure to side-step the general-case hardness result of [4].

Related work on kinetic simulators. There are a variety of simulators suited to strand-displacement systems and self-assembly based systems. One goal is to scale up to large scaffold lengths, both in 1D and 2D (with pseudoknots), which may require new features not seen in existing fine/coarse-grained strand displacement simulators [25, 2, 19, 23, 32]. Also, our use of toehold mediated strand displacement, as well as future goals to handle displacement through helices (in 2D), require features that go beyond existing self-assembly simulators [30, 9, 11]. Our kinetic model uses rates from the peppercorn reaction enumerator [2]. In future work we could use oxDNA [25] to further inform and calibrate rates.

1.2 Future work

Are there fast thermodynamic prediction algorithms for other engineered multistranded and/or pseudoknotted systems? Specifically, for DNA strand displacement circuits, or DNA tile-based self-assembly systems, or even DNA origami systems? Our main motivation for this question, and in fact for this work, is the idea that designed systems and abstractions might side-step known hardness results and imply the existence of efficient MFE or partition function algorithms. Here, for 1D SDC, two things helped: (1) using a domain-based model, justified by enforcement of domain-level abstraction/interactions with careful sequence design, and (2) that the set of all 1D SDC configurations have an implicit structure amenable to divide-and-conquer, despite that set being exponentially large in system size.

Continuing this reasoning a step further, one may ask, beyond the existence of a fast, polynomial time, thermodynamic prediction algorithm, can one attempt to classify engineered DNA systems based on the computational complexity of their MFE or partition function? Theorem 3 shows that MFE for 1D SDC is in the complexity class L(with the assumption $k = O(1)$; we leave it open to remove/generalise that caveat). Hence, we can say that if we encode the output of a computation in an MFE configuration then the model solves only problems in L. We know that (general) multi-stranded systems are NP-complete MFE [4]. Here, we are asking if there are natural systems with MFE that lie in other complexity classes, for example perhaps the wide variety of engineered domain-based DNA systems have algorithmic hardness of MFE that are characterised by some of the many complexity classes within P [16, 14].

More concretely, it remains open to characterise the complexity of predicting, reasonable domain-level formalisations of, 2D SDC systems.

2 1D Scaffolded DNA Computer (SDC): model definition

Previous work [24] defined the SDC at tile, domain, and sequence levels of abstraction, but in a mainly experimental context. In this section we formally define the 1D SDC at the domain level of abstraction, with sufficient extra technical detail to facilitate our proofs.

2.1 A simple domain-based model of 1D SDC

▶ **Definition 4** (domain, complement, strand, binding)**.** A *domain* d is an ordered triple $(\text{name}, \text{len}, \text{dir})$, where name is a string over a fixed alphabet, $\text{len} \in \mathbb{N}$ is the length of d, and $\text{dir} \in \{\rightarrow, \leftarrow\}$ is the domain direction. The complement of domain d is the equal-length opposite-direction domain $\bar{d}$. An n-domain strand s, is an ordered n-tuple of domains $s = (d_1, d_2, ..., d_n)$ all of the same direction (the strand is said to have that direction). We say $d_i < d_j$ if $i < j$. Two strands s_1 and s_2 may bind at domain d, if one has domain d and the other $\bar{d}$.

▶ **Definition 5** (1D Scaffolded DNA Computer (SDC))**.** A 1D Scaffolded DNA Computer (Figure 2), $\mathcal{S}$, is an ordered pair (S, T) where $S = (d_1, d_2, \ldots d_N)$ is an N-domain *scaffold strand* with direction $\leftarrow$, with all N *scaffold domains* being distinct. T is a set of 3-domain *computation strands*, of direction $\rightarrow$, and each $s \in T$ is the form $s = (d^{\mathrm{L}}, d_i^{\mathrm{M}}, d^{\mathrm{R}})$ where $d^{\mathrm{L}}, d_i^{\mathrm{M}}, d^{\mathrm{R}}$ are the domains on left, middle and right of s respectively, and where d_i^{M} binds to a scaffold domain d_i and $d^{\mathrm{L}}, d^{\mathrm{R}}$ are called *computation domains* and they do not bind to any scaffold domain. $\mathcal{S}$ is said to be of length N.

We say a computation strand s *competes* at scaffold domain d_i if its middle domain $d_i^{\mathrm{M}} = \bar{d}_i$. The set of d_i*-competing strands* is $C_{d_i} = \{s \mid s = (d^{\mathrm{L}}, d_i^{\mathrm{M}}, d^{\mathrm{R}}) \in T \text{ and } d_i^{\mathrm{M}} = \bar{d}_i\}$. Since scaffold domains are distinct, if $d_i \neq d_j$ then $C_{d_i} \cap C_{d_j} = \emptyset$ (hence each computation strand binds at most at one scaffold domain). Sometimes, the first scaffold domain, d_1, is called the input domain. For any two computation strands $s \in C_{d_i}$ and $s' \in C_{d_j}$, we say $s \prec s'$ if $d_i < d_j$.

▶ **Definition 6** (SDC configuration, SDC secondary structure)**.** For a 1D SDC $\mathcal{S}$ of length N, a *configuration* X is a sequence of $l \leq N$ computation strands $X = (s_1, s_2, \ldots, s_l)$ such that $s_i \prec s_{i+1}$. X is said to be of size l. We interpret a configuration as a *domain-level secondary structure* by binding up all adjacent matching domains, more formally for all $i \in [1, l]$:

(a) we bind the middle domain d_i^{M} of s_i to s_i's associated scaffold domain (since $s_i \in C_d$ for some d), and

(b) if $d_i^{\mathrm{R}} = \bar{d}_{i+1}^{\mathrm{L}}$ and s_i, s_{i+1} are adjacent on the scaffold, i.e. if the right domain of s_i is complement to the left of s_{i+1} and if the scaffold indices of s_i, s_{i+1} differ by exactly 1, we bind those two domains.

In the previous definition, the condition $s_i \prec s_{i+1}$ ensures that a configuration has at most one competing computation strand per scaffold domain.

A *computation* is a sequence of configurations $\epsilon \vdash X_1 \vdash X_2 \vdash \ldots \vdash X_t$ that begins with the empty configuration ϵ (corresponding to a scaffold with nothing bound to it). A computation step, $X_j \vdash X_{j+1}$, is where a computation strand s binds at some scaffold domain d, where s is non-deterministically chosen from those d-competing strands C_d that preserve or increase the total number of bound domains.[3] A configuration X_j is *final* if X_j has the maximum number of bound domains out of all possible configurations. A final configuration may be unique or not.[4]

Figure 2(c) shows an initial configuration (left), and two final configurations (right). A useful intuition for computation in this model: the selection of a strand (from the finite set of possibilities for that position) executes a logical step, and step-by-step the system will moves towards an enthalpically favourable reachable configuration (or a cycle of them).

2.2 Ensemble of secondary structures

An MFE or partition function algorithm computes over an ensemble (i.e. a set) of permissible secondary structures. In this work, we choose that ensemble to be the set of domain-level secondary structures, that each include a scaffold strand with zero or more computation strands bound to that scaffold and also to each other, or more formally:

[3] Note that off-scaffold interactions are forbidden in this formalism. In Section 2.3 we will define a free-energy based condition that more closely matches experimental implementation.

[4] In the Bit-Copy example in Figure 2 the system eventually reaches one of two polymers (all 0 or all 1), depending on the input (0 or 1). Other example SDC systems have multiple equal-energy polymers, and there may or may not be a cycle (of length > 1) of steps between elements of a set of polymers. A cycle can happen where there are several polymers (each with an identical number of unbound domains) and the system transitions between these.

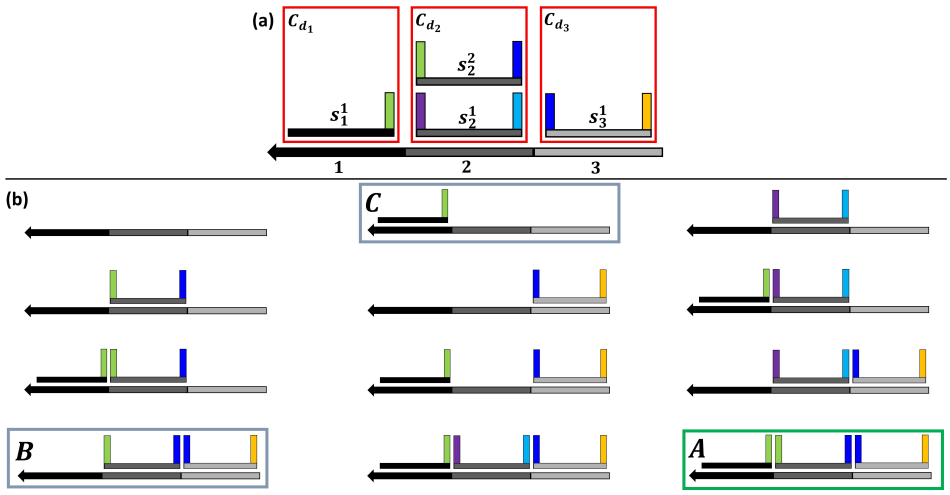


Figure 3 A small example 1D SDC and its ensemble of configurations. (a) An example 1D SDC $\mathcal{S}$ of length 3 showing its scaffold and four computation strands. At scaffold positions 1, 2 and 3 there are respectively 1, 2 and 1 competing computation strands. (b) The ensemble of configurations $\Omega^{\mathcal{S}}$, with A highlighted as the unique MFE configuration of this system (bound domains everywhere).

▶ **Definition 7** (ensemble $\Omega^{\mathcal{S}}$)**.** The ensemble $\Omega^{\mathcal{S}}$ of a 1D SDC $\mathcal{S}$ is the set of all configurations of $\mathcal{S}$ interpreted as domain-level secondary structures (see Definition 6).

For a 1D SDC $\mathcal{S}$ of length N that has $k_1, k_2, ..., k_N$ computation strands competing at the N scaffold domains, we have $|\Omega^{\mathcal{S}}| = (k_1 + 1)(k_2 + 1) \cdots (k_N + 1)$ since each scaffold domain d_i has k_i possible strands that bind it plus the possibility of it being unbound. If there are exactly k competing strands at each scaffold domain, $|\Omega^{\mathcal{S}}| = (k + 1)^N$.

▶ **Example 8.** Figure 3 shows a simple 1D SDC of length 3 and its ensemble of twelve configurations/domain-level secondary structures.

Our choice of secondary structure ensemble is justified as follows: We ignore off-scaffold interactions by making the assumption that the 1D SDC system is designed so that (a) scaffold and computational strands do not self-bind, (b) any pair of matching computation domains are sufficiently weak that a pair of 3-domain strands are unlikely to bind for much time (even if they bind on two computation domains), (c) by using a principle analogous to DNA origami [21], we assume the scaffold is at low concentration relative to the computation strands and that (d) scaffold binding domains are strong ensuring that binding to the scaffold is highly favoured.

It can be seen that the domain-level secondary structures we consider are unpseudoknotted. This fact is not required to prove our theorems, so we omit (the straightforward) proof.

2.3 Energy model

We augment each domain d to have an associated negative real-valued free energy $\Delta G(d) \in \mathbb{R}, \Delta G(d) < 0$. Also, we define ΔG^{assoc} to be a strictly positive real number such that $-\Delta G(d) > \Delta G^{\text{assoc}}$. For notation, we let $\Delta G(d, d') = \Delta G(d)$ if $d = \bar{d'}$, and $\Delta G(d, d') = 0$ otherwise.

For a 1D SDC $\mathcal{S}$ of length N, the free energy of a configuration $X \in \Omega^{\mathcal{S}}$ of size l is:

$$\Delta G^{\mathcal{S}}(X) = \sum_{s \in X} \Delta G(d^{\text{M}}(s)) + l \cdot \Delta G^{\text{assoc}} + \sum_{s_i, s_{i+1} \in X} \Delta G(d^{\text{R}}(s_i), d^{\text{L}}(s_{i+1})). \quad (1)$$

where the latter summation sums the free energy of bound *scaffold-adjacent* strand pairs. Note that since X represents $l+1$ strands *including the scaffold*, the strand association penalty of $l \cdot \Delta G^{\text{assoc}}$ is consistent with previous work on multiple strands, e.g. [7, 4]). The 1D SDC domain-level MFE and partition function are respectively:

$$\text{MFE}^{\mathcal{S}} = \min_{X \in \Omega^{\mathcal{S}}} \{\Delta G^{\mathcal{S}}(X)\} \tag{2}$$

$$Q^{\mathcal{S}} = \sum_{X \in \Omega^{\mathcal{S}}} e^{-\Delta G^{\mathcal{S}}(X)/k_{\text{B}}T} \tag{3}$$

▶ **Example 9.** The free energy of configuration B in Figure 3 is:

$$\Delta G^{\mathcal{S}}(B) = \Delta G(d^{\text{M}}(s_2^2)) + \Delta G(d^{\text{M}}(s_3^1)) + 2\Delta G^{\text{assoc}} + \Delta G(d^{\text{R}}(s_2^2), d^{\text{L}}(s_3^1))$$

Configuration A in Figure 3 has MFE of all configurations, $\text{MFE}^{\mathcal{S}} = \Delta G^{\mathcal{S}}(A)$, since A has all scaffold domains bound, and all adjacent computation domains bound.

3 Thermodynamics: polynomial time 1D SDC MFE algorithm

Before proving the main result of this section, Theorem 1, we give some preliminary results.

3.1 Definitions for combining configurations & ensemble partitioning

▶ **Definition 10** ($X \circledast Y$)**.** For any two configurations X and Y such that $\forall s \in X, \forall s' \in Y$ such that $s \in C_d \implies s' \notin C_d$, we say X is compatible with Y. The interlacing of two compatible configurations X and Y is the configuration $X \circledast Y = (s_1, s_2, \ldots, s_k)$ such that $s_i \prec s_{i+1}$, and either $s_i \in X$ or $s_i \in Y$. The interlacing of a configuration X with the set of configurations Λ where X is compatible with Λ, is the set $X \circledast \Lambda = \{X \circledast Y \mid \text{ for each } Y \in \Lambda\}$.

▶ **Example 11.** Configuration B is compatible with configuration C in Figure 3 since they do not share any common, bound scaffold domain, $B \circledast C = A$.

The following definitions will be useful to efficiently partition the (exponentially large) SDC ensemble $\Omega^{\mathcal{S}}$ to compute its MFE and partition function.

▶ **Definition 12** ($\Omega_s^{\mathcal{S}}$)**.** For a 1D SDC $\mathcal{S}$, $\Omega_s^{\mathcal{S}}$ denotes the set of all configurations having the computation strand s as the (rightmost) strand, more formally, $\Omega_s^{\mathcal{S}} = \{X \mid X \in \Omega^{\mathcal{S}}$ and $s \in X$ and $\forall s' \in X$ such that $s' \neq s \implies s' \prec s\}$.

The intuition behind Lemma 13 comes from: if we have any configuration with final strand s, we can see it as the interlacing of two smaller configurations one of the two is just $\{s\}$ itself and the other ends with a final strand $s' \prec s$.

▶ **Lemma 13.** *In any 1D SDC $\mathcal{S}$, for any computation stand s, $\Omega_s^{\mathcal{S}} = [s \circledast \bigcup_{s' \prec s} \Omega_{s'}^{\mathcal{S}}] \bigcup \{s\}$ and $|\Omega_s^{\mathcal{S}}| = 1 + \sum_{s' \prec s} |\Omega_{s'}^{\mathcal{S}}|$.*

▶ **Definition 14** (Ω-associated MFE)**.** For a set of configurations $\Omega \subset \Omega^{\mathcal{S}}$ of a 1D SDC $\mathcal{S}$, the Ω-associated MFE, $\text{MFE}^{\mathcal{S}}(\Omega) = \min_{X \in \Omega}\{\Delta G^{\mathcal{S}}(X)\}$. If $\Omega = \Omega_s^{\mathcal{S}}$ for some computation strand s, then for simplicity we will denote $\text{MFE}^{\mathcal{S}}(\Omega_s^{\mathcal{S}})$ by $\text{MFE}_s^{\mathcal{S}}$.

▶ Remark 15. To simplify our proofs, from Observation 17 onwards, for any scaffold domain d, we assume that $|C_d| \neq 0$, meaning there is at least one computation strand competing at d. At the end of this section (Remark 20) we will show how our proposed algorithm can be easily used to overcome this restriction.

From Section 2.3, (a) all domain binding energies are negative, and (b) $\Delta G(s) < -\Delta G^{\text{assoc}}$ for a scaffold-bound strand s. Also, for a configuration Y of length $l < N$ we can add $N - l$ computation strands to the $N - l$ unbound scaffold positions, lowering the free energy (this fact is used in the following two lemmas). The following lemma follows from the inclusion $\Omega^{\mathcal{S}}_{s'} \subset \Omega^{\mathcal{S}}_{s}$:

▶ **Lemma 16.** *In any 1D SDC $\mathcal{S}$, for any two different computation strands s and s' such that if $s' \prec s$, then $MFE^{\mathcal{S}}_{s'} > MFE^{\mathcal{S}}_{s}$.*

▶ **Observation 17.** *For a 1D SDC $\mathcal{S}$ of length N, there is a configuration X of size N such that $MFE^{\mathcal{S}} = \Delta G^{\mathcal{S}}(X)$.*

▶ **Lemma 18.** *In any 1D SDC $\mathcal{S}$, for any two different computation strands s and s' such that if $s' \prec s$, then $MFE^{\mathcal{S}}(\Omega^{\mathcal{S}}_{s'} \circledast \{s\}) = MFE^{\mathcal{S}}(\Omega^{\mathcal{S}}_{s'}) + \Delta G^{\mathcal{S}}(d^{\text{M}}(s)) + \Delta G^{\text{assoc}} + \Delta G(d^{\text{R}}(s'), d^{\text{L}}(s))$.*

3.2 Polynomial time 1D SDC MFE algorithm

The main result of this section is the following (restated) theorem:

▶ **Theorem 1.** There is an $O(k^2N)$ time algorithm to determine the domain-level MFE for a 1D SDC of length N with $\leq k$ computation strands competing at each scaffold domain.

Proof. We will prove that Algorithm 1, intuitively illustrated in Appendix A, returns the recursively-defined quantity $M^{\mathcal{S}}$ defined in Equation (4), and does so in $O(k^2N)$ steps (we assume the standard RAM model [6] for algorithm analysis and $\Delta G = O(1)$ per domain).

$$M^{\mathcal{S}} = \min_{s \in C_{d_N}} \{M^{\mathcal{S}}_s\} \tag{4}$$

$$M^{\mathcal{S}}_s = \Delta G(d^{\text{M}}(s)) + \Delta G^{\text{assoc}} + \min_{s' \in L_s} \{M^{\mathcal{S}}_{s'} + \Delta G(d^{\text{R}}(s'), d^{\text{L}}(s))\} \tag{5}$$

where C_{d_N} is the set of strands competing at the final/rightmost scaffold domain (Section 2.1), L_s is the set of strands that bind to the domain immediately "to the left" of s on the scaffold (i.e. if $s \in C_{d_i}$ then $L_s = C_{d_{i-1}}$, and where $L_s = \emptyset$ if $s \in C_{d_1}$), and the remaining notation is given in Section 2.3.

To prove the time bound: The for loops in Lines 3 and 5 iterate over all strands s in the system, iteratively implementing the recursion in Equation (5), with each iteration (for s) representing a corresponding L_s (Equation (5)). Then the min over s' (Equation (5)) is executed by Line 10 (as a min over m). There are $\leq Nk$ strands, and the min requires $O(k)$ steps, giving $O(Nk^2)$ time. Finally, Line 14 executes Equation (4) in an additional $O(k)$ steps, yielding the claim.

It remains to show that $M^{\mathcal{S}}$ (Equation (4)) equals $\text{MFE}^{\mathcal{S}}$ (see Equation (2)): By Lemma 19 (below), for any computation strand s, $M^{\mathcal{S}}_s = \text{MFE}^{\mathcal{S}}_s$, thus it is immediate that the same strand s satisfies both $\min_{s \in T} M^{\mathcal{S}}_s = \min_{s \in T} \text{MFE}^{\mathcal{S}}_s$. Since an MFE configuration has size N, there is some $s \in C_{d_N}$, based on the assumption in Remark 15, such that $\text{MFE}^{\mathcal{S}}_s = \text{MFE}^{\mathcal{S}}$, and hence for that s, $M^{\mathcal{S}}_s = \text{MFE}^{\mathcal{S}}$. ◀

▶ **Lemma 19.** *For any 1D SDC $\mathcal{S}$, for any computation strand s, $M^{\mathcal{S}}_s = MFE^{\mathcal{S}}_s$ (where $M^{\mathcal{S}}_s$ is from Equation (5) and $MFE^{\mathcal{S}}_s$ is from Definition 14).*

Proof. We will prove the statement by induction on x, such that x is the domain index where the computation strand s is competing. Formally if $s \in C_{d_i}$, then $x = i$.

Algorithm 1 1D SDC MFE algorithm. The proof of Theorem 1 proof shows that this algorithm returns the value $M^{\mathcal{S}}$ defined in Equation (4). Note that arrays are indexed from 1. Notation: $k_1, \ldots, k_N$ are the counts of competing strands at scaffold domains $d_1, \ldots, d_N$. Let s_i^j be the j^{th} strand competing at domain d_i.

$M_{\text{curr}} = [0, 0, \ldots, 0]$ ▷ size $k = \max(k_1, \ldots, k_N)$ for current MFEs
$M_{\text{prev}} = [0, 0, \ldots, 0]$ ▷ size $k = \max(k_1, \ldots, k_N)$ for previous MFEs
for $i \leftarrow 1 \ldots N$ **do** ▷ index scaffold domains
  $M_{\text{prev}} \leftarrow M_{\text{curr}}$
  for $j \leftarrow 1 \ldots k_i$ **do** ▷ index computational strands at scaffold domain d_i
    if $i = 1$ **then** ▷ first scaffold domain, has no left neighbour
      $M_{\text{curr}}[j] \leftarrow \Delta G(d^{\text{M}}(s_i^j))$
    else
      ▷ $O(k)$ steps to choose min and bind scaffold + entropic penalty
      $M_{\text{curr}}[j] \leftarrow \left[\min_{m \in \{1,2,\ldots,k_{i-1}\}} \left(M_{\text{prev}}[m] + \Delta G\left(d^{\text{R}}(s_{i-1}^m), d^{\text{L}}(s_i^j)\right)\right) + \Delta G(d^{\text{M}}(s_i^j)) + \Delta G^{\text{assoc}}\right]$
    end if
  end for
end for
$M^{\mathcal{S}} \leftarrow \min_{k' \in \{1,2,\ldots,k_N\}} M_{\text{curr}}[k']$ ▷ $O(k)$ steps implement Equation (4) giving $M^{\mathcal{S}}$
return $M^{\mathcal{S}}$

Base step: if $x = 1$, then this means that s is a strand competing at the first scaffold domain, then using Lemma 13 there is only one configuration $X \in \Omega_s^{\mathcal{S}}$, and $X = s$. Then we have $M_s^{\mathcal{S}} = \Delta G(d^{\text{M}}(s)) + \Delta G^{\text{assoc}} = \Delta G^{\mathcal{S}}(X) = \text{MFE}_s^{\mathcal{S}}$, and this completes the base step.

Induction step: assume the induction hypothesis is valid for all $x < y$, in other words that $M_{s'}^{\mathcal{S}} = \text{MFE}_{s'}^{\mathcal{S}}$ (the former from Equation (5), the latter from Definition 14) because x, y are domain indices and $s' \prec s$.

Then, if $x = y$, we can replace $M_{s'}^{\mathcal{S}}$ in Equation (5) by $\text{MFE}_{s'}^{\mathcal{S}}$, we get the following: $M_s^{\mathcal{S}} = \Delta G(d^{\text{M}}(s)) + \Delta G^{\text{assoc}} + \min_{s' \in L_s}\{\text{MFE}_{s'}^{\mathcal{S}} + \Delta G(d^{\text{L}}(s),\ d^{\text{R}}(s'))\}$. As $\Delta G(d^{\text{M}}(s)) + \Delta G^{\text{assoc}}$ is just a constant, and using the basic properties of the min operator, we get the following: $M_s^{\mathcal{S}} = \min_{s' \in L_s}\{\text{MFE}_{s'}^{\mathcal{S}} + \Delta G(d^{\text{M}}(s)) + \Delta G^{\text{assoc}} + \Delta G(d^{\text{L}}(s),\ d^{\text{R}}(s'))\}$. Using Lemma 18, we will get the following: $M_s^{\mathcal{S}} = \min_{s' \in L_s}\{\text{MFE}^{\mathcal{S}}(\Omega_{s'}^{\mathcal{S}} \circledast \{s\})\}$. Using Definition 14, we get: $M_s^{\mathcal{S}} = \min_{s' \in L_s}\{\min_{X \in (\Omega_{s'}^{\mathcal{S}} \circledast \{s\})}\{\Delta G^{\mathcal{S}}(X)\}\}$. For each distinct computation strand s', it is easy to show that each set of configurations $[\Omega_{s'}^{\mathcal{S}} \circledast \{s\}]$ is disjoint. Let $\Omega' = \bigcup_{s' \in L_s} [\Omega_{s'}^{\mathcal{S}} \circledast \{s\}]$, so again we use another property of the min operator to get the following: $M_s^{\mathcal{S}} = \min_{X \in \Omega'}\{\Delta G^{\mathcal{S}}(X)\}$. From Observation 17 we know that if Y is the configuration that has the MFE of the partition $\Omega_s^{\mathcal{S}}$, then Y must be of length l if $s \in C_{d_l}$, in other words Y must have computation strands only on scaffold domains $0, 1 \ldots, l$. Any configuration of size l that ends in s is contained in $\Omega_s^{\mathcal{S}}$, and is also contained in Ω', and hence $Y \in \Omega'$, and we get that $M_s^{\mathcal{S}} = \min_{X \in \Omega'}\{\Delta G^{\mathcal{S}}(X)\} = \min_{X \in \Omega_s^{\mathcal{S}}}\{\Delta G^{\mathcal{S}}(X)\}$. So $M_s^{\mathcal{S}} = \min_{X \in \Omega_s^{\mathcal{S}}}\{\Delta G^{\mathcal{S}}(X)\} = \text{MFE}_s^{\mathcal{S}}$. ◀

▶ Remark 20. Now, we can easily remove the assumption in Remark 15 by dividing the scaffold into pieces separated by domains d such that $C_d = \emptyset$ and running MFE on each such region, and then summing each MFE.

Finally, Appendix D gives an analysis of Algorithm 1 that yields the proof of Theorem 3.

Algorithm 2 1D SDC partition function algorithm. The proof of Theorem 2 argues that this algorithm returns $Z^{\mathcal{S}}$ as defined in Equation (6). Note that arrays are indexed from 1, and recall that $k_1, \ldots, k_N$ are the counts of competing strands at scaffold domains $d_1, \ldots, d_N$, and we let s_i^j be the j^{th} strand competing at domain d_i. See Figure 9.

$Z_{\text{curr}} = [0, 0, \ldots, 0]$ ▷ size $k = \max(k_1, \ldots, k_N)$, current (partial) partition function
$Z_{\text{prev}} = [0, 0, \ldots, 0]$ ▷ size $k = \max(k_1, \ldots, k_N)$, previous (partial) partition function
$Z^{\mathcal{S}} \leftarrow 1$; $\text{sum}_a \leftarrow 0$
for $i \leftarrow 1 \ldots N$ **do**
  $\text{sum}_a \leftarrow \text{sum}_a + \sum_{i \in \{1,\ldots,k\}} Z_{\text{prev}}[i]$ ▷ sum_a: rightmost summation Equation (7)
  $Z_{\text{prev}} \leftarrow Z_{\text{curr}}$
  $Z_{\text{curr}} = [0, 0, \ldots, 0]$
  for $j \leftarrow 1 \ldots k_i$ **do** ▷ each iteration computes Equation (7) for a strand
    $t_1 = \mathrm{e}^{-(\Delta G(d^{\mathrm{M}}(s_i^j)) + \Delta G^{\text{assoc}})/k_{\mathrm{B}}T}$
    if $i = 1$ **then** ▷ first domain where is no neighbors at all
      $Z_{\text{curr}}[j] = t_1$
    else
      $t_2 \leftarrow 0$
      for $m \leftarrow 1 \ldots k_{i-1}$ **do**
        $t_2 \leftarrow t_2 + \left(\mathrm{e}^{-\left(\Delta G(d^{\mathrm{R}}(s_{i-1}^m), d^{\mathrm{L}}(s_i^j))\right)/k_{\mathrm{B}}T}\right) \cdot Z_{\text{prev}}[m]$
      end for
      $Z_{\text{curr}}[j] \leftarrow t_1 + t_2 + \text{sum}_a$
    end if
    $Z^{\mathcal{S}} \leftarrow Z^{\mathcal{S}} + Z_{\text{curr}}[j]$ ▷ computing Equation (6)
  end for
end for
return $Z^{\mathcal{S}}$

4 Thermodynamics: polynomial time 1D SDC partition function

▶ **Definition 21** (Ω-associated partition function)**.** For a 1D SDC $\mathcal{S}$, for any set of configurations $\Omega \subset \Omega^{\mathcal{S}}$, the Ω-associated partition function is $Q^{\mathcal{S}}(\Omega) = \sum_{X \in \Omega} e^{-\Delta G^{\mathcal{S}}(X)/k_{\mathrm{B}}T}$. If $\Omega = \Omega_s^{\mathcal{S}}$ for some computation strand s, then for simplicity we will denote $Q^{\mathcal{S}}(\Omega_s^{\mathcal{S}})$ by $Q_s^{\mathcal{S}}$.

4.1 A polynomial time partition function algorithm for 1D SDC

We restate the main result of this section:

▶ **Theorem 2.** There is an $O(k^2 N)$ time algorithm for the domain-level partition function for a 1D SDC of length N with $\leq k$ computation strands competing at each scaffold domain.

Proof. We will first claim that Algorithm 2, intuitively illustrated in Figure 9 (Appendix B), returns the recursively-defined quantity $Z^{\mathcal{S}}$ (Equation (6)), and does so in $O(k^2 N)$ steps:

$$Z^{\mathcal{S}} = 1 + \sum_{s \in T} Z_s^{\mathcal{S}} \tag{6}$$

$$Z_s^{\mathcal{S}} = \left(e^{-(\Delta G(d^{\mathrm{M}}(s)) + \Delta G^{\text{assoc}})/k_{\mathrm{B}}T}\right) \cdot \left[1 + \sum_{s' \in L_s} Z_{(s',s)} * Z_{s'}^{\mathcal{S}} + \sum_{s' \prec s \text{ and } s' \notin L_s} Z_{s'}^{\mathcal{S}}\right] \tag{7}$$

where $Z_{(s',s)} = e^{-\Delta G(d^{\mathrm{R}}(s'),d^{\mathrm{L}}(s))/k_{\mathrm{B}}T}$. Here, $Z_s^{\mathcal{S}}$ is an intermediate quantity (that we will prove is the Ω_s-associated partition function), L_s is the set of strands that bind to the domain immediately "to the left" of s on the scaffold (i.e. if $s \in C_{d_i}$ then $L_s = C_{d_{i-1}}$, and where $L_s = \emptyset$ if $s \in C_{d_1}$).

To see the claim note that Lines 4 and 8 iterate over all $s \in T$ ($O(kN)$ iterations), and each inner-loop iteration calculates $Z_s^{\mathcal{S}}$ for some strand $s \in T$. Line 14 for loop iterates $O(k)$ times and computes the left-hand summation of Equation (7) (giving $O(k^2N)$ total steps), and Line 5 computes the right-hand summation. Line 19 accumulates the value of $Z^{\mathcal{S}}$ at each iteration, and when the algorithm terminates $Z^{\mathcal{S}}$ is equal to Equation (6).

Lemma 23 below proves that $Z^{\mathcal{S}} = Q^{\mathcal{S}}$, which completes the proof. ◀

▶ **Lemma 22.** *For any 1D SDC $\mathcal{S}$, for any computation strand s, $Z_s^{\mathcal{S}} = Q_s^{\mathcal{S}}$ (where $Z_s^{\mathcal{S}}$ is from Equation (7) and $Q_s^{\mathcal{S}}$ is from Definition 21)*

Proof. We will prove it by induction on scaffold domain index x, with the induction hypothesis being that $Z_s^{\mathcal{S}} = Q_s^{\mathcal{S}}$ for all s at scaffold domain index $\leq x \in [1, N]$.

Base step, $x = 1$: This implies that strand s is competing at first scaffold domain d_1, then using Lemma 13 there is only one secondary structure $X \in \Omega_s^{\mathcal{S}}$, hence $X = s$, a configuration of size 1. Then we compete the base step via:

$$Z_s^{\mathcal{S}} = \mathrm{e}^{-(\Delta G(d^{\mathrm{M}}(s)) + \Delta G^{\mathrm{assoc}})/k_{\mathrm{B}}T} = \mathrm{e}^{-(\Delta G^{\mathcal{S}}(X))/k_{\mathrm{B}}T} = Q_s^{\mathcal{S}}.$$

We re-write $Z_s^{\mathcal{S}}$ (Equation (7)) as follows:

$$Z_s^{\mathcal{S}} = \left(\mathrm{e}^{-(\Delta G(d^{\mathrm{M}}(s)) + \Delta G^{\mathrm{assoc}})/k_{\mathrm{B}}T}\right) \cdot \left[1 + \sum_{s' \prec s} Z_{(s',s)} \cdot Z_{s'}^{\mathcal{S}}\right]. \tag{8}$$

which is equivalent to Equation (7), as $Z_{(s',s)} = 1$ if $s' \notin L_s$ since $\Delta G(d^{\mathrm{R}}(s'), d^{\mathrm{L}}(s))$ will equal 0 as s', s are not adjacent, i.e. there is no interaction between them.

Induction step: assume the induction hypothesis is valid for all $x < y$, then if $x = y$, we can replace $Z_{s'}^{\mathcal{S}}$ in Equation (8) by $\sum_{X \in \Omega_{s'}^{\mathcal{S}}} e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}} = Q_s^{\mathcal{S}}$ (Definition 21), we get

$$Z_s^{\mathcal{S}} = \left(\mathrm{e}^{-\frac{\Delta G(d^{\mathrm{M}}(s)) + \Delta^{assoc}}{k_{\mathrm{B}}T}}\right) \cdot \left[1 + \sum_{s' \prec s} Z_{(s,s')} \cdot \sum_{X \in \Omega_{s'}^{\mathcal{S}}} e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}}\right]. \tag{9}$$

Using the distributive property of multiplication over addition we get

$$Z_s^{\mathcal{S}} = \mathrm{e}^{-\frac{\Delta G(d^{\mathrm{M}}(s)) + \Delta G^{\mathrm{assoc}}}{k_{\mathrm{B}}T}} + \sum_{s' \prec s} \sum_{X \in \Omega_{s'}^{\mathcal{S}}} \left[\mathrm{e}^{-\frac{\Delta G(d^{\mathrm{M}}(s)) + \Delta G^{\mathrm{assoc}}}{k_{\mathrm{B}}T}} \cdot Z_{(s,s')} \cdot e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}}\right]. \tag{10}$$

We then apply Lemma 25 since these $\Omega_{s'}^{\mathcal{S}}$ sets, for different s' are disjoint by Lemma 24, then we can replace the double summation by only one over $\bigcup_{s' \prec s} \Omega_{s'}^{\mathcal{S}}$ and we get

$$Z_s^{\mathcal{S}} = e^{-\frac{\Delta G(d^{\mathrm{M}}(s)) + \Delta G^{\mathrm{assoc}}}{k_{\mathrm{B}}T}} + \sum_{X \in \bigcup_{s' \prec s} \Omega_{s'}^{\mathcal{S}}} \left[e^{-\frac{\Delta G(d^{\mathrm{M}}(s)) + \Delta G^{\mathrm{assoc}}}{k_{\mathrm{B}}T}} \cdot Z_{(s',s)} \cdot e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}}\right]. \tag{11}$$

We can rewrite Equation (11) as follows:

$$Z_s^{\mathcal{S}} = \sum_{X \in \left[\{\epsilon\} \cup \bigcup_{s' \prec s} \Omega_{s'}^{\mathcal{S}}\right]} \left[e^{-\frac{\Delta G(d^{\mathrm{M}}(s)) + \Delta G^{\mathrm{assoc}}}{k_{\mathrm{B}}T}} \cdot Z_{(s',s)} \cdot e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}}\right]. \tag{12}$$

The latter transition is justified because if $X = \epsilon$, then $Z_{(s',s)} = 1$ and $\Delta G^{\mathcal{S}}(X) = 0$ since there are no computation strand s' to interact with s as X is the empty configuration.

Let $\Omega_1 = [\{\epsilon\} \cup \bigcup_{s' \prec s} \Omega^{\mathcal{S}}_{s'}]$, $\Omega_2 = \Omega^{\mathcal{S}}_s$, $f : \Omega_1 \to \mathbb{R}$, $f' : \Omega_1 \to \mathbb{R}$, $g : \Omega_2 \to \mathbb{R}$ and $g' : \Omega_2 \to \mathbb{R}$. f', f, g' and g are defined as follows:

- $f'(X) = \Delta G^{\mathcal{S}}(X) + \Delta G(d^{\mathrm{M}}(s)) + \Delta G(d^{\mathrm{R}}(s_l), d^{\mathrm{L}}(s)) + \Delta G^{\mathrm{assoc}}$,
- $f(X) = e^{-f'(X)}$,
- $g'(X') = \Delta G^{\mathcal{S}}(X')$, and
- $g(X') = e^{-g'(X')}$,

where X is a configuration of length l and $s_l \in X$ is the final (right-most) strand on the scaffold of configuration X (which may be at some domain index $\leq N$). We know from Observation 27 that there is a direct one-to-one correspondence, $\alpha : \Omega_1 \to \Omega_2$, defined by $\alpha(X) = s \circledast X$. Notice that function f' is defined in a way that computes the free energy, Equation (1), of the configuration that results from $X' = \alpha(X)$, so $f'(X) = g'(\alpha(X)) = g'(X')$. It directly follows that $f(X) = e^{-f'(X)} = e^{-g'(X')} = g(X')$, then we can apply Lemma 26 directly on Equation (12) using the previous setup for that lemma and we will get $Z^{\mathcal{S}}_s = \sum_{X \in \Omega^{\mathcal{S}}_s} e^{-\Delta G^{\mathcal{S}}(X)/k_{\mathrm{B}}T}$, and this completes the proof. ◀

▶ **Lemma 23.** *For any 1D SDC $\mathcal{S}$, $Z^{\mathcal{S}} = Q^{\mathcal{S}}$, where $Z^{\mathcal{S}}$ is defined in Equation (6) and $Q^{\mathcal{S}}$ is the partition function of $\mathcal{S}$ (Equation (3)).*

Proof. Since $Z^{\mathcal{S}} = 1 + \sum_{s \in T} Z^{\mathcal{S}}_s$, and from Lemma 22, we know that $Z^{\mathcal{S}}_s = \sum_{X \in \Omega^{\mathcal{S}}_s} e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}}$.

Then, we get the following: $Z^{\mathcal{S}} = 1 + \sum_{s \in T} \sum_{X \in \Omega^{\mathcal{S}}_s} e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}}$. Again we can apply Lemma 25 directly because these $\Omega^{\mathcal{S}}_s$ sets, for different s, are disjoint from Lemma 24, then we can replace the double summation by only one over $\bigcup_{s \in T} \Omega^{\mathcal{S}}_s$, and with the aid of Observation 28, we get the following:

$$Z^{\mathcal{S}} = 1 + \sum_{X \in \bigcup_{s \in T} \Omega^{\mathcal{S}}_s} e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}} = 1 + \sum_{X \in \Omega^{\mathcal{S}} \setminus \{\epsilon\}} e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}} = \sum_{X \in \Omega^{\mathcal{S}}} e^{-\frac{\Delta G^{\mathcal{S}}(X)}{k_{\mathrm{B}}T}}.$$ ◀

5 Kinetics: A simple domain-based kinetic model of 1D SDC

In this section, we first propose a simple kinetic model for the 1D SDC. Second, we implement a simulator for this model (in java and python languages, as a Gillispie simulation [13]) and use it to evaluate existing systems. Third, we use the model to evaluate two hypotheses for improving the kinetics of *constant temperature* 1D SDC systems.

Model. Our kinetic model for a 1D SDC, $\Theta = (S_N, T)$, is a continuous-time Markov chain (CTMC) that satisfies detailed balance [23]. The initial system state as shown in Figure 4(c) is the empty configuration, ϵ, then the simplest version of our kinetic model assumes that the first N steps are merely the binding of N computation strands to the scaffold (no displacement). This assumption is reasonable since we know empirically that hybridization reaction rates are much faster than reactions utilising strand displacement (this assumption can be removed, but we leave it for simplicity). Then any next step will be a sub-step of either toehold exchange or strand displacement, based on a 3-step and 2-step model, respectively described in the following equations and shown in Figure 4(a):

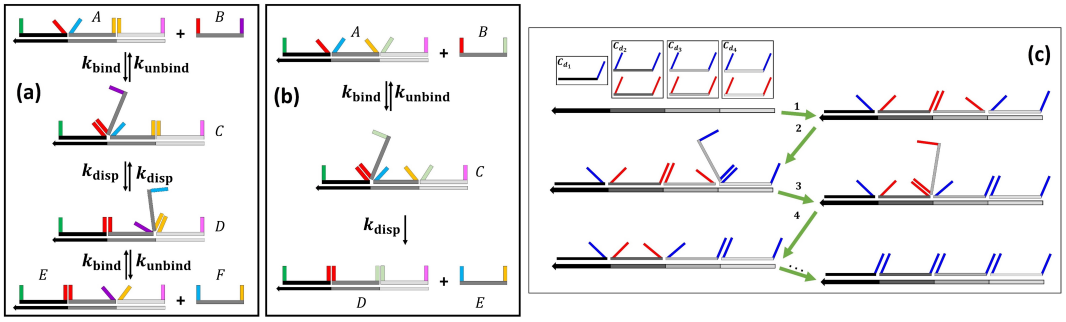


Figure 4 Summary of kinetic 1D SDC model that extends the thermodynamic model in Section 2.1 (and Figure 2). (a) 3 step bind-displace-unbind. (b) 2 step bind-displace. (c) An example computation. We start with the all-empty scaffold, after some fast hybridization events, we have a random scaffold configuration whereupon a sequence of strand displacement events eventually yield the unique thermodynamically-favoured target configuration.

$$\mathrm{A} + \mathrm{B} \underset{\mathrm{k_{unbind}}}{\overset{\mathrm{k_{bind}}}{\rightleftharpoons}} \mathrm{C} \underset{\mathrm{k_{disp}}}{\overset{\mathrm{k_{disp}}}{\rightleftharpoons}} \mathrm{D} \underset{\mathrm{k_{bind}}}{\overset{\mathrm{k_{unbind}}}{\rightleftharpoons}} \mathrm{E} + \mathrm{F}$$

$$\mathrm{A} + \mathrm{B} \underset{\mathrm{k_{unbind}}}{\overset{\mathrm{k_{bind}}}{\rightleftharpoons}} \mathrm{C} \xrightarrow{\mathrm{k_{disp}}} \mathrm{D} + \mathrm{E}$$

where A, B, C, D, E, F are complex names, and the binding and unbinding rates, r_{bind} and r_{unbind} respectively, are $r_{\text{bind}} = [s] * k_{\text{bind}}$ and $r_{\text{unbind}} = k_{\text{unbind}} * e^{(\Delta G(d^{\text{R}}(s'), d^{\text{L}}(s)) + \Delta G^{\text{assoc}})/k_{\text{B}}T}$, where s, s' are computation strands with matching computation domains, $[s]$ denotes the concentration of s and k_{disp}, k_{bind} and k_{unbind} denote the (single, global) displacement, binding, and unbinding rate, respectively. We used the peppercorn enumerator [2] and NUPACK [7] to obtain these rates.

Simulator. Our simulator for this model works as follows: If we assume the system is currently in state x, the rate for each possible next state x' will be computed, the sum of these rates will be used as a normalization factor to compute a probability for each such next state. Then we compute cumulative density function (CDF) of the random variable that is the next state, and uniformly generate a random number from the interval $[0, 1]$ and use it to determine which next state, x', the system will go to. This process will be repeated forever until the system goes to a halt state, if there is one (which is a state with no next states).

Motivation for two proposals. An interesting challenge for 1D SDC goes thus: suppose we are in the typical situation where our 1D SDC program has a single computation strand, d_1, competing at the first scaffold domain and several computation strand competing at the rest of scaffold domains, and that we begin in an otherwise random configuration. On a large scaffold, there will be many mismatching computational domains, implying many competing random walks (moving to the left and right), and we will in the worst case need to wait something like quadratic time in scaffold length for the single strand at d_1 to "win" all competitions leading to the most enthalpically favoured state. The question we ask here: are there any tricks that we can use to speed up the kinetics?

Note that in this situation the input computation strand, i.e. the leftmost computation strand, plays a crucial role both in determining the target configuration, and in driving the system to that target. Hence, our intuition is that there is value in considering kinetic tricks that push (speed-up) computation in the left-to-right direction. Below, we propose two tricks that build on this intuition.

Figure 5 Proposal 1: Scaffold with covers (here at the second, fourth, and sixth scaffold domains).

Proposal 1: Covers. A *cover* is simply a strand consisting of a single domain $\bar{d}$ which is complementary to some scaffold domain d, such that when the cover is at high concentration relative to computation strands that compete at domain d, the cover out-competes the computation strand. If we have covers for every scaffold domain, the scaffold gets coated, or mostly coated, in covers as shown in Figure 5. The key idea here is that covers have no computation domains, and hence do not send left or right randomly-walking signals, intuitively making covers faster to displace. We carried out a number of simulations with various cover excesses. Figure 7 shows the results (scaffold is at 0.1x, computation strands at 1x, and covers at higher excesses).

Proposal 2: Monotonically increasing computation strand concentrations. The intuition here is that by having higher concentrations of computation strands at higher scaffold domain indices (i.e. as we move further from the start position we increase computation strand concentration) the random walk should be biased to move to the right at a higher rate, than with all equal concentrations. The idea is illustrated in Figure 6: First imagine that we have a Bit-Copy system, such that at each scaffold position i we have two computation strands s_i^0, s_i^1 (copying bit 0 or 1, respectively). We set concentrations to be strictly increasing with respect to i, in other words $[s_i^0] = [s_i^1] \leq [s_{i+1}^0] = [s_{i+1}^1]$. Thus, for any choice of bit $b \in \{0, 1\}$, the strand s_{i+1}^b has an equal or higher binding rate than s_i^b. Figure 7 shows result with scaffold at 0.1x, and where a computation strand at scaffold position i has concentration i x.

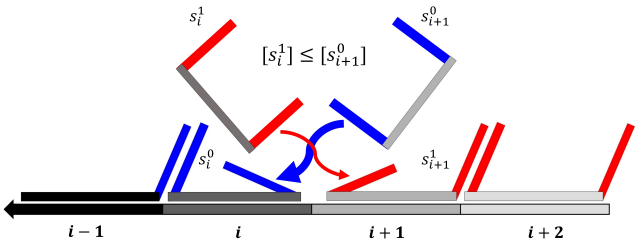


Figure 6 Proposal 2: increasing concentration from left to right. Computation strands further to the right (higher subscript) have higher concentrations than those to the left.

References

1 Tatsuya Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104(1-3):45–62, 2000.

2 Stefan Badelt, Casey Grun, Karthik V Sarma, Brian Wolfe, Seung Woo Shin, and Erik Winfree. A domain-level DNA strand displacement reaction enumerator allowing arbitrary non-pseudoknotted secondary structures. *Journal of the Royal Society Interface*, 17(167):20190866, 2020.

3 Harry M. T. Choi, Maayan Schwarzkopf, Mark E. Fornace, Aneesh Acharya, Georgios Artavanis, Johannes Stegmaier, Alexandre Cunha, and Niles A. Pierce. Third-generation in situ hybridization chain reaction: multiplexed, quantitative, sensitive, versatile, robust. *Development*, 145(12):dev165753, June 2018.

4 Anne Condon, Monir Hajiaghayi, and Chris Thachuk. Predicting minimum free energy structures of multi-stranded nucleic acid complexes is APX-hard. In Matthew Lakin and Petr

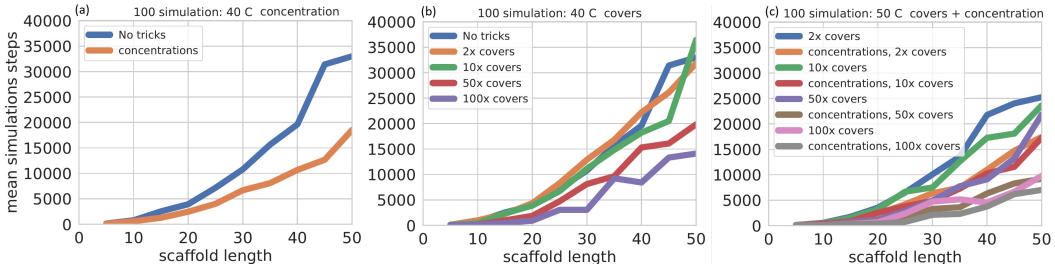


Figure 7 1D SDC kinetics simulations for the Bit-Copy program. 100 simulations were run for each scaffold length that is a multiple of 5 (from 5 to 50), until the target (output) configuration was reached, with the mean number of simulation steps for those 100 shown. (a) The blue curve shows the basic kinetic model (scaffold at 0.1x, computation strands at 1x). Orange shows results for Proposal 2 where scaffold position i has computation strand concentration i x, showing some speed-up. (b) Proposal 1: Covers. Simulations show that high cover concentrations lead to significant speed-ups. (c) Mixing both proposals leads to greater speedups than either.

Šulc, editors, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2021. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

5 Matthew Cook, Tristan Stérin, and Damien Woods. Small tile sets that compute while solving mazes. In Matthew Lakin and Petr Šulc, editors, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–20, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. Arxiv preprint: `arXiv:2106.12341`.

6 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 2022.

7 Robert M Dirks, Justin S Bois, Joseph M Schaeffer, Erik Winfree, and Niles A Pierce. Thermodynamic analysis of interacting nucleic acid strands. *SIAM review*, 49(1):65–88, 2007.

8 David Doty, Trent A Rogers, David Soloveichik, Chris Thachuk, and Damien Woods. Thermodynamic binding networks. In *DNA23: The 23rd International Conference on DNA Computing and Molecular Programming*, volume 10467 of *LNCS*, pages 249–266. Springer, 2017.

9 Constantine G Evans. Rgrow tile self-assembly simulator, May 2023. `doi:10.5281/zenodo.7915489`.

10 Constantine G Evans, Jackson O'Brien, Erik Winfree, and Arvind Murugan. Pattern recognition in the nucleation kinetics of non-equilibrium self-assembly. *arXiv preprint arXiv:2207.06399*, 2022.

11 Constantine G Evans and Erik Winfree. Physical principles for DNA tile self-assembly. *Chemical Society Reviews*, 46(12):3808–3829, 2017.

12 Cody Geary, Paul WK Rothemund, and Ebbe S Andersen. A single-stranded architecture for cotranscriptional folding of RNA nanostructures. *Science*, 345(6198):799–804, 2014.

13 Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

14 Raymond Greenlaw, H James Hoover, and Walter L Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, USA, 1995.

15 Rune B Lyngsø and Christian NS Pedersen. RNA pseudoknot prediction in energy-based models. *Journal of computational biology*, 7(3-4):409–427, 2000.

16 Cristopher Moore and Stephan Mertens. *The Nature of Computation*. Oxford University Press, 2011.

17 Ruth Nussinov, George Pieczenik, Jerrold R Griggs, and Daniel J Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied mathematics*, 35(1):68–82, 1978.

18 Christos H Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

19 Andrew Phillips and Luca Cardelli. A programming language for composable DNA circuits. *Journal of the Royal Society Interface*, 6(suppl_4):S419–S436, 2009.

20 Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *science*, 332(6034):1196–1201, 2011.

21 Paul WK Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

22 John SantaLucia Jr and Donald Hicks. The thermodynamics of DNA structural motifs. *Annu. Rev. Biophys. Biomol. Struct.*, 33:415–440, 2004.

23 Joseph Malcolm Schaeffer, Chris Thachuk, and Erik Winfree. Stochastic simulation of the kinetics of multiple interacting nucleic acid strands. In *DNA Computing and Molecular Programming: 21st International Conference, DNA 21, Boston and Cambridge, MA, USA, August 17-21, 2015. Proceedings 21*, pages 194–211. Springer, 2015.

24 Tristan Stérin, Abeer Eshra, and Damien Woods. Thermodynamically favoured algorithms on a scaffolded DNA computer. In preparation. (Preliminary version presented at The 28th International Conference on DNA Computing and Molecular Programming (DNA28), Track B, 2022.).

25 Petr Šulc, Flavio Romano, Thomas E Ouldridge, Lorenzo Rovigatti, Jonathan PK Doye, and Ard A Louis. Sequence-dependent thermodynamics of a coarse-grained DNA model. *The Journal of chemical physics*, 137(13):135101, 2012.

26 Boya Wang, Cameron Chalk, and David Soloveichik. SIMD||DNA: Single instruction, multiple data computation with DNA strand displacement cascades. In *DNA Computing and Molecular Programming: 25th International Conference, DNA 25, Seattle, WA, USA, August 5–9, 2019, Proceedings 25*, pages 219–235. Springer, 2019.

27 Boya Wang, Siyuan Stella Wang, Cameron Chalk, Andrew Ellington, and David Soloveichik. Parallel molecular computation on digital data stored in DNA. *bioRxiv*, pages 2022–08, 2022.

28 Michael S Waterman and Temple F Smith. RNA secondary structure: A complete mathematical analysis. *Mathematical Biosciences*, 42(3-4):257–266, 1978.

29 Bryan Wei, Mingjie Dai, and Peng Yin. Complex shapes self-assembled from single-stranded DNA tiles. *Nature*, 485(7400):623–626, 2012.

30 Erik Winfree, Rebecca Schulman, and Constantine Evans. The xgrow simulator, 2003.

31 Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. *Nature*, 567(7748):366–372, 2019.

32 Sedigheh Zolaktaf, Frits Dannenberg, Erik Winfree, Alexandre Bouchard-Côté, Mark Schmidt, and Anne Condon. Efficient parameter estimation for DNA kinetics modeled as continuous-time markov chains. In *DNA Computing and Molecular Programming: 25th International Conference, DNA 25, Seattle, WA, USA, August 5–9, 2019, Proceedings 25*, pages 80–99. Springer, 2019.

33 Michael Zuker and Patrick Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981.

A Appendix: MFE algorithm illustration

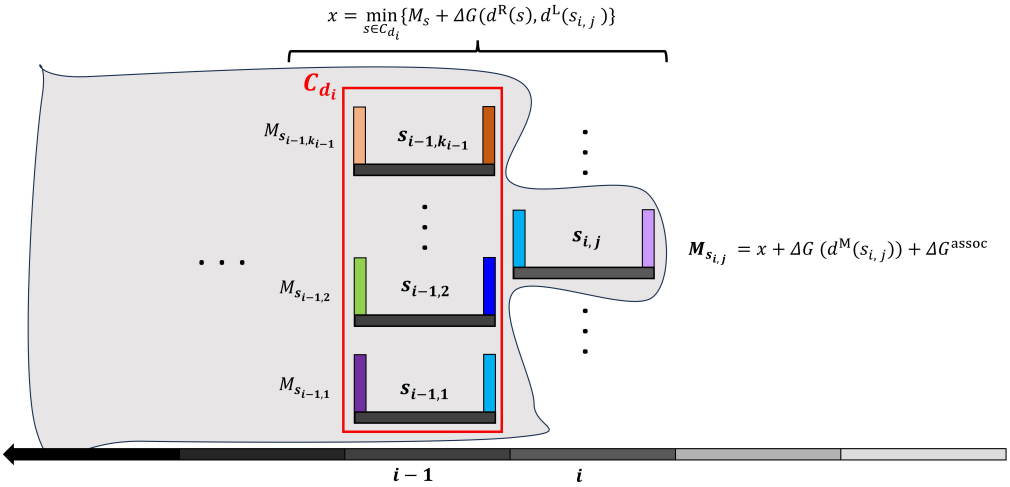


Figure 8 Graphical illustration of Algorithm 1 (and for the proof of Theorem 1). The algorithm iterates through scaffold domains, with i being the current scaffold domain. The strand $s_{i,j}$ is the j^{th} strand (out of k_i strands) at scaffold domain i. Strand $s_{i,j}$ has an associated "partial" MFE denoted $M_{s_{i,j}}$ which is the MFE of all configurations that have strand $s_{i,j}$ as their rightmost strand (and have no strands binding to scaffold domains $d_{>i}$). Algorithm 1 proceeds, for each of the $\leq k_i$ strands s at scaffold position i, by populating the list M_{curr}, the j^{th} entry of which will contain the computed value $M_{s_{i,j}}$. In order to compute $M_{s_{i,j}}$, Algorithm 1 makes use of a list M_{prev} that maintains a similar list for index $i-1$, and merely needs to take a min over entries of M_{prev} added to the interaction with computation strand $s_{i,j}$ (shown as the equation for x at the top).

B Appendix: Partition function algorithm illustration

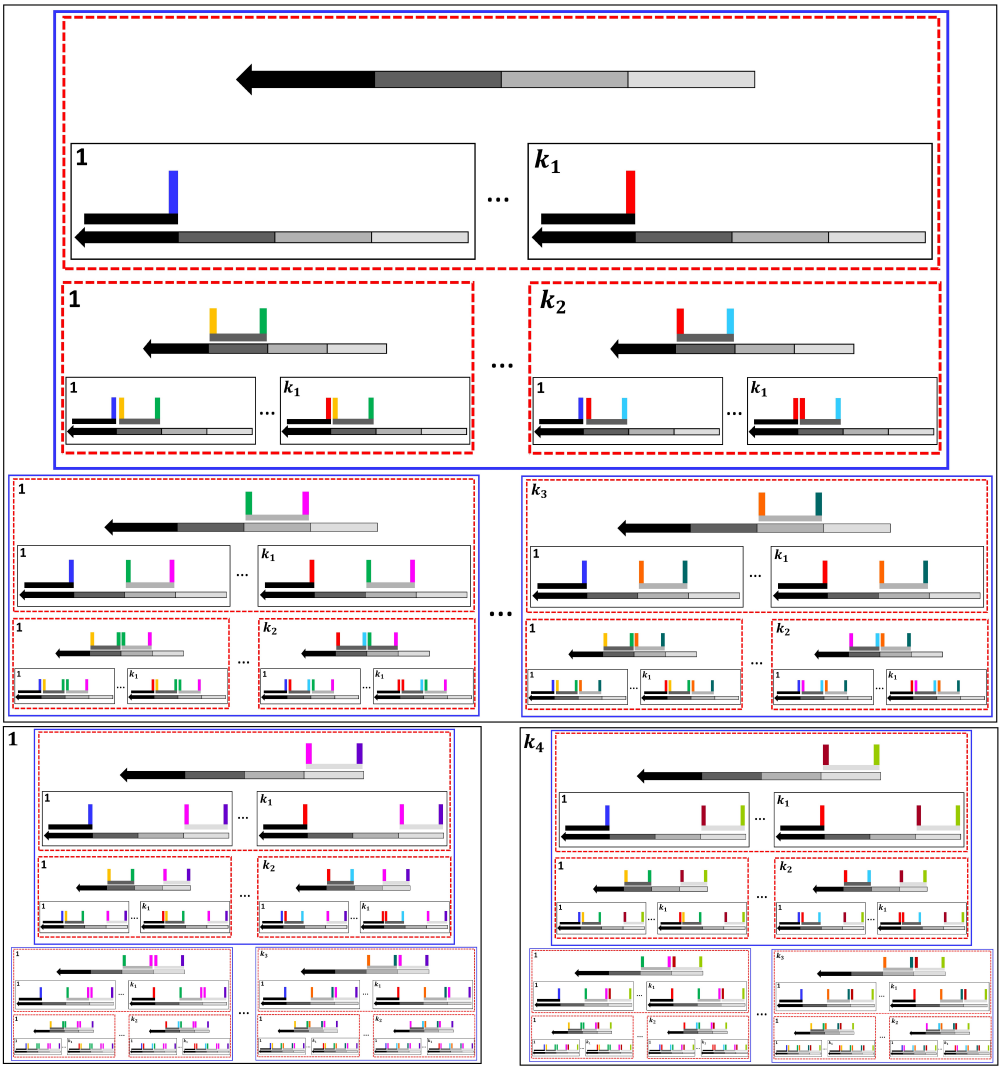


Figure 9 Graphical illustration for Algorithm 2 (and Theorem 2). The coloured boxes contain sets of configurations. The recursive structure in the figure corresponds to the recursive structure of Equation (7): for example, the top blue box corresponds to the summation of the Ω_s-associated partition function for the following partitions: ϵ (empty scaffold on top), $\Omega^{\mathcal{S}}_{s_1^1}, \Omega^{\mathcal{S}}_{s_1^2}, \ldots, \Omega^{\mathcal{S}}_{s_1^{k_1}}$ (top red box), $\Omega^{\mathcal{S}}_{s_2^1}, \Omega^{\mathcal{S}}_{s_2^2}, \ldots, \Omega^{\mathcal{S}}_{s_2^{k_2}}$ (lower level k_2 red boxes in top blue box). Recursively, the computation represented by the top blue box, is then used to compute the partition function for all strands s that compete at domain d_3, and so on.

C Thermodynamics: Basic lemmas

Here, we give some lemmas, proofs are straightforward and hence omitted. Lemma 24 gives a simple method to partition the 1D SDC ensemble $\Omega^{\mathcal{S}}$ using the identity of the rightmost (final) strand (the proof is immediate from the two strands s_1, s_2 being distinct, giving the

required partition). The proof of Lemma 25 follows from the associative property of addition and the disjointedness of the involved collection of configuration sets, and Lemma 26 follows from existence of the one-to-one mapping α and how it is defined.

▶ **Lemma 24.** *In any 1D SDC $\mathcal{S}$, for any two distinct computation strands $s_1 \neq s_2$, $\Omega^{\mathcal{S}}_{s_1} \bigcap \Omega^{\mathcal{S}}_{s_2} = \emptyset$.*

▶ **Lemma 25.** *In any 1D SDC $\mathcal{S}$, for any finite collection of disjoint sets of configurations $\Omega_1, \Omega_2, ..., \Omega_n$ and any real-valued function $f : \Omega^{\mathcal{S}} \to \mathbb{R}$,*

$$\sum_{i=1}^{n} \sum_{X \in \Omega_i} f(X) = \sum_{X \in \Omega'} f(X) \text{ where } \Omega' = \bigcup_{i=1}^{n} \Omega_i$$

▶ **Lemma 26.** *In any 1D SDC $\mathcal{S}$, for any two finite sets of configurations Ω_1, Ω_2 and for any two real-valued functions $f : \Omega_1 \to \mathbb{R}$ and $g : \Omega_2 \to \mathbb{R}$ such that there is a one-to-one correspondence, α, between Ω_1 and Ω_2, and $f(X) = g(\alpha(X))$, then $\sum_{X \in \Omega_1} f(X) = \sum_{X' \in \Omega_2} g(X')$.*

▶ **Observation 27.** *As a direct consequence from Lemma 13, for any 1D SDC $\mathcal{S}$, and for any computation strand s, we define a one-to-one correspondence $\alpha : \{\epsilon\} \bigcup_{s' \prec s} \Omega^{\mathcal{S}}_{s'} \to \Omega^{\mathcal{S}}_{s}$, by interlacing the strand s to each configuration X that belongs to the domain of α, $\alpha(X) = X \circledast s$.*

▶ **Observation 28.** *As an important special case of Lemma 13, for any 1D SDC $\mathcal{S}$ of length N, we can show the following $\Omega^{\mathcal{S}}$ partitioning: $\Omega^{\mathcal{S}} = \left[\bigcup_{s \in T} \Omega^{\mathcal{S}}_{s}\right] \bigcup \{\epsilon\}$. By ordering computation strands in T based on their domain indices, we can view $\Omega^{\mathcal{S}}$ as union of a hierarchy of partitions beginning with $\{\epsilon\}$ then all $\Omega^{\mathcal{S}}_{s}$ for all $s \in C_{d_1}$ and so on until all $\Omega^{\mathcal{S}}_{s}$ for all $s \in C_{d_N}$.*

D Computational complexity of MFE

We restate and prove the following theorem. Note that MFE values are ≤ 0.

▶ **Theorem 3.** Given a 1D SDC $\mathcal{S}$ of length $N \in \mathbb{N}$, with $k = O(1)$ computing strands per scaffold domain and bounded domain energies ($\Delta G(d) \in O(1)$, for a domain d binding to its complement), and a finite-precision decimal r, the problem of deciding whether the domain-level MFE of $\mathcal{S}$ is $\leq r$ is in the complexity class L.

Proof. We assume familiarity with deterministic logarithmic space Turing machines and the class L [16, 18]. Let $n = \max(N, \text{len}(r))$ where $\text{len}(r)$ is the number of decimal symbols used to specify r, ignoring leading and trailing 0s. First, we argue that all variables in Algorithm 1 fit in $O(\log n)$ workspace: The MFE value $M^{\mathcal{S}}$ produced by the algorithm is a negative number bounded below by $M \stackrel{\text{def}}{=} N \cdot (\Delta G(d^M(s)) + \Delta G^{\text{assoc}}) + (N-1) \cdot \Delta G(d^{\text{R}}(s'))$, where s is the strand with the minimum ΔG (strongest) middle domain d^M, and s' is the strand with the minimum ΔG (strongest) right domain d^{R} (i.e. we are taking a min over all domains of the system to compute a putative free energy that no configuration can be less than). Hence $M \in O(N) = O(n)$, and thus $M \leq M^{\mathcal{S}}$ can be written down using $O(\log n)$ bits, keeping in mind that these quantities are all ≤ 0. M_{curr} and M_{prev} in the algorithm are lists of length $k \in O(1)$, and no value stored in these lists is less than M (since M is a lowerbound for the MFE). Finally, the various operations in the algorithm (minimum of a list, addition, assignment, and iteration) are all logspace computable. Hence our MFE problem is in L. ◀

E Application of our partition function algorithm

Armed with a fast partition function algorithm (Algorithm 2) we can analyse 1D SDC systems. Figure 2(d) shows an example result, where we used our partition function algorithm to "simulate" an anneal of the Bit-Copy system, for scaffold lengths up to 400 (i.e. 801 strands), by computing the probability of the target configuration (correct output) at every 2° C from 100° C down to 20° C.[5] We used temperature-dependent domain binding energies computed from DNA sequences from ongoing experimental work [24] (mean ΔG of 24-base scaffold and 12-base computational domains being -20.7 and -11.2 kcal/mol for intended binding, respectively, at 55° C).

For non-complementary (orthogonal) computational domains at adjacent positions on the scaffold, we allowed some binding free-energy (16% of intended binding, an approximation derived from the NUPACK-computed partition function free-energy for a designed sequence set [24]), and a positive (unfavourable) penalty for two bound computational domains (to approximate geometric strain of three helices in close proximity/loop closure). With these minor tweaks, the probability of the target (correct Bit-Copy output) with respect to temperature showed qualitative agreement with experimental results on an $N = 4$ 1D SDC.

Our partition function algorithm allows for analysis of much longer scaffold lengths, useful for future scaled-up designs, with Figure 2(d) showing scaffold lengths up to 400 (i.e. Bit-Copy with 801 strands). This test was meant as a speed test for our algorithm, and not a quantitative prediction of the experimental system, which we leave to future work. In particular there remains some work to calibrate our domain-level model to experiments. For example, lowering the sequence orthogonality constraint from 16% to 8% increases the predicted melting temperature by ≤ 2 degrees for scaffold length ≤ 100, implying that there is some, albeit low, sensitivity to crucial, and difficult to measure, system parameters.

F Source code

Source code for our algorithms can be found at `https://dna.hamilton.ie/software.html`.

[5] An earlier, brute-force exponential-time algorithm could not go beyond scaffold lengths ≥ 15 on our hardware). Note that the target (output) is the structure with MFE.

DNA Tile Self-Assembly for 3D-Surfaces: Towards Genus Identification

Florent Becker
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Shahrzad Heydarshahi
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Abstract

We introduce a new DNA tile self-assembly model: the Surface Flexible Tile Assembly Model (SFTAM), where 2D tiles are placed on host 3D surfaces made of axis-parallel unit cubes glued together by their faces, called polycubes. The bonds are flexible, so that the assembly can bind on the edges of the polycube. We are interested in the study of SFTAM self-assemblies on 3D surfaces which are not always embeddable in the Euclidean plane, in order to compare their different behaviors and to compute the topological properties of the host surfaces.

We focus on a family of polycubes called *order-1 cuboids*. *Order-0 cuboids* are polycubes that have six rectangular faces, and order-1 cuboids are made from two order-0 cuboids by substracting one from the other. Thus, order-1 cuboids can be of genus 0 or of genus 1 (then they contain a tunnel). We are interested in the genus of these structures, and we present a SFTAM tile assembly system that determines the genus of a given order-1 cuboid. The SFTAM tile assembly system which we design, contains a specific set Y of tile types with the following properties. If the assembly is made on a host order-1 cuboid C of genus 0, no tile of Y appears in any producible assembly, but if C has genus 1, every terminal assembly contains at least one tile of Y.

Thus, for order-1 cuboids our system is able to distinguish the host surfaces according to their genus, by the tiles used in the assembly. This system is specific to order-1 cuboids but we can expect the techniques we use to be generalizable to other families of shapes.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Tile self-assembly, DNA computing, Geometric surfaces

Digital Object Identifier 10.4230/LIPIcs.DNA.29.2

Related Version *Full Version*: `https://arxiv.org/abs/2210.13094`

Funding Supported by ANR project SyDySi (ANR JCJC 2019 19-CE48-0007-01).

1 Introduction

In this paper, we introduce a new tile self-assembly model in order to perform self-assembly on 3-dimensional surfaces. The field of tile self-assemby originates in the work of Wang [14], who introduced in 1961 *Wang tiles*, that is, equally sized 2-dimensional unit squares with labels/colors on each edge (later called *glues*) and designed a Turing universal computation model based on these tiles. In 1998, inspired by Wang tiles and DNA complexes from Seeman's laboratory [6], Winfree introduced in his PhD thesis [15] the *abstract Tile Assembly Model* (aTAM). This model uses Wang tiling with an extra information: he associated a non-negative integer strength for each glue label. The power of DNA self-assembly enables to compute using this model. We refer to the survey [9] for more details on the literature, and to the online bibliography of Seeman's laboratory [12].

Most of the early work in the DNA tile self-assembly literature deals with rigid assemblies in the Euclidean plane [9, 10] (since the assemblies are discrete, the Euclidean plane is usually seen as the lattice $\mathbb{Z}^2$), which is a natural and simple setting for this model. However, it can be interesting to use self-assembly in richer settings. This has been investigated

experimentally for instance in [13, 16, 17] where the assembly takes place on a preexisting surface and changes according to the surface. On the theoretical side, there have been some recent works on DNA tile self-assembly outside the Euclidean plane, such as tile self-assembly in mazes [4], where the tile placement is done on the walls of a certain maze. Other types of self-assembly exist that also do not use the Euclidean plane, for example a model of cross-shaped origami tiles [18]. Another type of self-assembly not in the plane is 3D assemblies of complex molecules like crystals [3, 7]. Inspired by this, a recent model called *Flexible Tile Assembly Model* (FTAM) was introduced by Durand-Lose et al. in 2020 [5], as an extension of earlier work [8]. Here, we have Wang tiles but they self-assemble (without an input surface) in 3D space (modeled by the lattice $\mathbb{Z}^3$) as they can have, in addition to standard *rigid* bonds, *flexible* bonds that allow tiles to bind at any angle along the tile edges. The goal of the FTAM model is to construct complex 3D structures called *polycubes* (3D shapes made of unit cubes) [2].

In 2010, Abel et al. [1] used a variant of the aTAM to implement shape replication, where tiles react to the shape of a preexisting pattern to reproduce it. The assembly takes place on the 1-dimensional border of a 2D pattern. Here, the main challenge is that the system must react to the shape of the space around, rather than to an external input it can read as it wants.

We are interested in studying what happens if we put the tiles on a given 3D surface, that is not necessarily topologically equivalent to the Euclidean plane. The intuition is that this could modify the computational behaviour of the tile self-assembly model, and we believe it will be interesting for practical systems, as in some practical settings, self-assembly could be performed on complex surfaces.

Inspired by the FTAM, we introduce a new model, called *Surface Flexible Tile Assembly Model* (SFTAM). In the SFTAM, we are given a 3D surface, on which the tiles of the self-assembly get placed. The SFTAM is an intermediate between aTAM and FTAM. Unlike in the FTAM, our aim for introducing the SFTAM is *not* for building 3D structures or surfaces: we assume that the host surface already exists. In the SFTAM, tile bonds are all flexible and the tiles can bind along the edges of the surface.

This model enables to use self-assembly on surfaces other than $\mathbb{Z}^2$. The aim of this article is to introduce the SFTAM model, and to demonstrate its usefulness by showing how it can be used on various types of surfaces. One of the most important properties of a surface is its *genus*, which, intuitively, is the number of "holes" in the surface. The Euclidean plane has genus 0. We are interested in using the SFTAM on surfaces with different values of genus. For that, we study the problem of characterizing the surface of the assembly, according to its genus, using the SFTAM. It is quit easy to devise a system which can behave in some way only on the torus, but it is harder to make sure that it has always this behavior when it is in fact on a torus.

We focus on a family of 3D surfaces called *cuboids*, which are special types of polycubes. Polycubes can form complex surfaces, and their genus can be high. We focus on a simple family of polycubes that can have genus 0 or genus 1. More specifically, we define an *order-0 cuboid* as a polycube which has only six faces. An *order-1 cuboid* $C_1 = C_0 \setminus C'_0$ is a polycube that is made from the difference of two order-0 cuboids C_0 and C'_0. Thus, an order-0 cuboid is a simple surface with genus 0, but an order-1 cuboid can either have genus 0 (potentially with a pit or concavity) or genus 1, if it has a tunnel.

In this paper, we will suppose that the SFTAM self-assembly is performed on the surface of an order-1 cuboid C. We design an SFTAM system whose assemblies differ when C is of genus 0 and of genus 1 and thus, can be used to detect the genus of the surface C of the

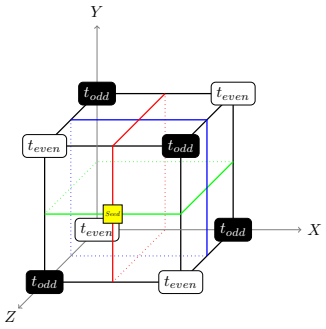


Figure 1 The skeleton of a $\mathcal{S}_\mathcal{G}$ assembly on an order-0 cuboid is shown in color. It is started from a seed (in yellow) and after the formation of the skeleton, the regions are partially filled by tiles of types t_{odd} and t_{even}.

assembly it is used on. The goal of this study is to show that performing self-assembly on surfaces of higher genus can be helpful. We also demonstrate some techniques which may prove useful in characterizing the topological properties of a wide range of surfaces.

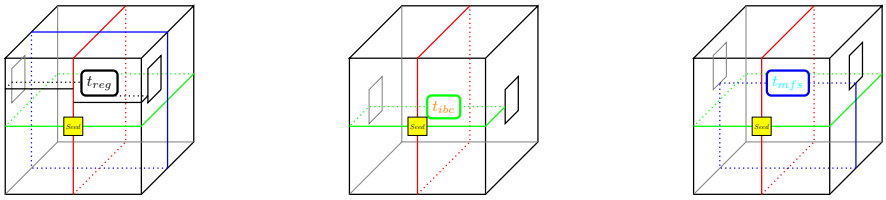
A *tile assembly system* (TAS) in the SFTAM is defined in a natural way as an extension of the aTAM: tile types are made of four glue labels, each has a strength, there is a seed assembly and a temperature (more formal definitions will be given later). An *assembly* is a placement of tiles on facets of the surface of the cuboid C. Two tiles bind if they are adjacent (i.e. their placements on the surface share an edge) and their glue labels are the same. In particular, edges are flexible and as a result the tiles can be placed on the border of orthogonal faces of C. The assembly is *producible* if it can be obtained by successfully binding tiles, starting from a seed . It is *terminal* if no additional tile can be bound to an existing tile.

Let $C = C_0 \setminus C'_0$ be an order-1 cuboid with its three dimensions at least 10 for C'_0. Our main result is to describe an SFTAM (TAS) $\mathcal{S}_\mathcal{G}$ with a subset Y of its tile types such that the following holds:

- if the order-1 cuboid C has genus 0, then no tile of Y appears in any producible assembly of $\mathcal{S}_\mathcal{G}$ on C, and
- if C has genus 1, every terminal assembly of $\mathcal{S}_\mathcal{G}$ on C contains at least one tile of Y.

In other words, the genus of C can be determined using the assemblies of $\mathcal{S}_\mathcal{G}$ on C. The assemblies of $\mathcal{S}_\mathcal{G}$ consist of two phases: a skeleton forms on the cuboid and separates it into several regions, then the regions are partially filled by inner tiles. See Fig. 1 for a sketch of the skeleton and its inner filling for an order-1 cuboid with genus 0. When the cuboid has genus 1, we show that there must be some parts of the skeleton or the inner filling which intersect in a way that is not possible on a genus-0 cuboid. The tile types of Y stick at the place where this happens. See Fig. 2.

We start with basic definitions and notations in Section 2, where we introduce and formalize our SFTAM model. Next, we introduce the family of order-1 cuboids and we show how the SFTAM behaves on the family of order-1 cuboids as an assembly model in three dimensions. In Section 3 we develop technical lemmas that will be necessary for the proof of our main result. In Section 4 we present our main result: a SFTAM tile assembly system that identifies the genus of order-1 cuboids using specific tiles from that system. We conclude in Section 5. Due to space constraints, some parts of the proofs and details are deferred to the appendix.



(a) The case where the skeleton does not meet the tunnel. In this case, tiles of types t_{odd} and t_{even} located in the two regions containing the entrances of the tunnel, pass inside the tunnel. Where they meet, a tile of type t_{reg} appears in the assembly.

(b) The tunnel intersects along the width of plane P_X and length of plane P_Y. The green tile is of type t_{ibc} (a tile type from T_{ibc}).

(c) The case where the tunnel of an order-1 cuboid is shown by a tile of type t_{mfs}, located at the intersection of the skeleton.

Figure 2 According to the relative position of the seed and the tunnel, the detection of the tunnel is done by different tile types of $\mathcal{S}_{\mathcal{G}}$. The seed is indicated in yellow and the skeleton is in color.

2 Definitions and notations

We now define the Surface Flexible Tile assembly Model, SFTAM. We work in 3-dimensional space, on the integer lattice $\mathbb{Z}^3$.

▶ **Definition 1** (Tile type in SFTAM). *Let Σ be a finite label alphabet and ϵ represent the null label. A* tile type *t is a 4-tuple $t = (t_1, t_2, t_3, t_4)$ with $t_i \in \Sigma \cup \{\epsilon\}$ for each $i = \{1, 2, 3, 4\}$. Each copy of a tile type is a* tile *and t_1, t_2, t_3, t_4 are the glues of t.*

Tiles are $2D$ unit squares whose sides are assigned the labels of the tile type. These squares are allowed to translate and rotate (unlike in aTAM), but they can not be mirrored (unlike in FTAM). In fact, since the tiles stick to a given surface, we can assume that they have an inner face and an outer face and that they always attach with the inner face in contact with the surface. In the definition of tile types, we show labels by numbers rather than cardinal directions. However, often, the orientation of a tile dictates the orientation of the tiles around it. Then, we use the expression "northern label" to refer to the label which will end up on the northern side (and similarly for east, west, south).

▶ **Definition 2** (Facet). *A* facet *is a face of the lattice $\mathbb{Z}^3$, i.e. a unit square whose vertices have integer coordinates.*

▶ **Definition 3** (Polycube). *A* polycube *is a 3D structure that is a subset of $\mathbb{Z}^3$ and is formed by the union of unit cubes that are attached by their faces.*

For a facet of a polycube, there are four possibilities for placing a tile.

▶ **Definition 4** (Placement). *Let C be a polycube. A* placement *$p = (f, o)$ on C consists of a facet f on the surface of C, and a side o of f, called its* orientation.

We denote the set of all placements in C by $Pl(C)$.

Given a tile type $t = (t_1, t_2, t_3, t_4)$ and a placement $p = (f, o)$, placing *t at the placement p defines a mapping from the edges of f to the label alphabet Σ. The i-th side of f (starting from the orientation o and going in clockwise direction, looking from the exterior of the surface of C) is associated with t_i.*

▶ **Definition 5** (Tile assembly system (TAS) on a polycube in SFTAM). *A* tile assembly system, *or TAS, over the surface of a given polycube C is a quintuple $\mathcal{S} = (\Sigma, T, \sigma, str, \tau)$, where :*

- *Σ is a finite label alphabet,*

- T *is a finite set of tile types on* Σ,
- σ *is called the* seed *and can be a single tile or several tiles*
- str *is a function from* $\Sigma \cup \{\epsilon\}$ *to non-negative integers called* strength function *such that* $str(\epsilon) = 0$, *and*
- $\tau \in \mathbb{N}$ *is called the* temperature.

While the SFTAM is a theoretical system, its components have an analogy with elements of practical DNA settings. The labels are the single strands of DNA, the function str show the strength of their connections and the τ is the temperature.

We present the definitions and notations of SFTAM assemblies that we will use throughout the article. They define similar to the ones for the aTAM [9].

▶ **Definition 6.** *An* assembly α *of a SFTAM TAS* $\mathcal{S}$ *on a polycube* C *is a partial function* $\alpha : \mathrm{Pl(C)} \dashrightarrow T$ *defined on at least one placement such that for each facet* f *of* C, *there is at most one placement* (f, o) *where* α *is defined.*

For placements $p = (f, o)$, $p' = (f', o')$ *of* $\mathrm{Pl(C)}$ *with* $\alpha(p) = t$ *and* $\alpha(p') = t'$ *such that* f *and* f' *are distinct but have a common side* s, *we say that* t *and* t' bind *together with the strength* st *if the glues of* t *and* t' *placed on* s *are equal and have the strength* st.

The assembly graph G_α *associated to* α *has as its vertices, the placements of* $\mathrm{Pl(C)}$ *that have an image by* α, *and two placements* p *and* p' *are adjacent in* G_α *if the tiles* $\alpha(p)$ *and* $\alpha(p')$ *bind.*

An assembly α *is* τ-stable *if for breaking* G_α *to any smaller assemblies, the sum of the strengths of disconnected edges of* G_α *needs to be at least* τ.

▶ **Definition 7.** *Let* C *be a polycube and* $\mathcal{S} = (\Sigma, T, \sigma, str, \tau)$ *a SFTAM TAS with* σ *positioned on a placement of* C. *An assembly* α *of* $\mathcal{S}$ *is* producible *on* C *if either* $dom(\alpha) = \{p\}$ *and* $\alpha(p) = \sigma$ *where* $p \in \mathrm{Pl(C)}$, *or if* α *can be obtained from a producible assembly* β *by adding a single tile from* $T \setminus \sigma$ *on* C, *such that* α *is* $\tau -$ *stable. Note that* $dom(\alpha)$ *is the domain of the assembly* α. *We denote the set of producible assemblies of* $\mathcal{S}$ *by* $A^C[\mathcal{S}]$. *An assembly is* terminal *if no tile can be* τ*-stably attached on* C. *The set of producible, terminal assemblies of* $\mathcal{S}$ *is denoted by* $A^C_\square[\mathcal{S}]$.

The SFTAM assemblies start from a seed and growth by a one by one tile adding. A tile can be added to an assembly in any placement on the host surface where it binds to the existing assembly with total strength at least τ with each pair of matching edges contributing the strength of its glue.

We now introduce *order-1 cuboids*, which are special types of polycubes.

▶ **Definition 8** (Order-0 cuboid)**.** *An* order-0 cuboid $C = (s_C, x_C, y_C, z_C)$ *where* $s_C = (s_x, s_y, s_z) \in \mathbb{Z}^3$ *is the point of* C *with smallest coordinates and* x_C, y_C, z_C *are integers representing the length, width and height of* C *is a 3D structure containing all points* (x, y, z) *of* $\mathbb{Z}^3$ *such that* $s_x \leq x \leq s_x + x_C$, $s_y \leq y \leq s_y + y_C$ *and* $s_z \leq z \leq s_z + z_C$. *We denote the set of all cuboids by* O_0.

We are interested in 3D structures that are more complicated than order-0 cuboids, in particular 3D structures that can have *tunnels*, that is, "holes".

▶ **Definition 9** (Order-1 cuboid)**.** *An* order-1 cuboid C_1 *is the difference between two elements of* O_0. *Given* $C_0 = (s_{C_0}, x_{C_0}, y_{C_0}, z_{C_0})$ *and* $C'_0 = (s_{C'_0}, x_{C'_0}, y_{C'_0}, z_{C'_0})$ *in* O_0. $C_1 = C_0 \setminus C'_0$ *is an order-1 cuboid if there is a* $i \in \{x, y, z\}$ *such that* $i_{C_0} \leq i_{C'_0}$. *We note* O_1 *the set of all order-1 cuboids.*

The genus of an order-1 cuboid is at most 1. The set of order-0 cuboids is a subset of the set of order-1 cuboids, that is, $O_0 \subseteq O_1$. An order-1 cuboid $C_1 = C_0 \setminus C_0'$ can be of three different types, depending on how C_0 and C_0' interact: (i) C_0 and C_0' have no intersection, and C_1 is an order-0 cuboid, (ii) C_0' cuts a hole in C_0, and C_1 is denoted as an order-1 cuboid with a *tunnel* and has genus 1; and (iii) C_0 and C_0' intersect but C_0' does not cut a hole in C_0. If the cut is in the inner face of C_0, C_1 is an order-1 cuboid with a *pit* and if the cut is in the side of a face of C_0, C_1 is an order-1 cuboid with a *concavity*. In both cases of order-1 cuboid with a pit or with a concavity, the genus is 0.

3 Finding the Middle of a Rectangle

Our main results uses some arithmetic and geometric computations, which are defined in $\mathbb{Z}^2$. A transfer lemma guarantees that they also work on any surface, if it is regular enough.

Given an assembly of SFTAM on $\mathbb{Z}^2$, the smallest axis-parallel rectangle containing the assembly is its *underlying rectangle*. If an SFTAM assembly is on a 3D surface, it is permitted to fold along the tiles' edges. The underlying rectangle is then the smallest subset of the surface which contains the assembly and is isomorphic to a rectangle of $\mathbb{Z}^2$, if it exists.

▶ **Lemma 10.** *Let α be a producible assembly of an SFTAM TAS S on $\mathbb{Z}^2$ with underlying rectangle R, and let C be a polycube. If there exists a function $i : \mathbb{Z}^2 \to C$ such that the restriction of i to R is a graph isomorphism, then the image of α under i is producible on C.*

Proof. If the seed is placed at p_s in $\mathbb{Z}^2$, it is placed at $i(p_s)$ on C. Since the tile bonds can fold along edges of C, the assembly on C proceeds exactly as in $\mathbb{Z}^2$, and each tile placed at a point p in $\mathbb{Z}^2$ is placed at point $i(p)$ on C. ◀

We design the following systems (details omitted due to space constraints) which are a variant of those of [11]. The IBC (Increasing Binary Counter) counts up to a number, while distinguishing rows which are a power of two. The DBC (Decreasing Binary Counter) counts down from a number, while distinguishing the row where 0 is reached. The U-Turn System makes a copy of a number from position $[(x, y), ..., (x + k, y)]$ to position $[(x - k - 1, y), ..., (x - 1, y)]$. See Figure 15 for an example of IBC and DBC Systems, and Figure 16 for U-Turn System.

An *explicitly bounded rectangle* R is a rectangle whose horizontal sides are bounded by specially marked tiles. The following lemma uses the IBC, DBC and U-Turn Systems in a SFTAM TAS that finds the middle of the height of R.

▶ **Lemma 11** (Middle finding system). *Let R be an explicitly bounded rectangle of height N and width at least $3\log(N)$. There is a TAS $S_{1/2} = (\Sigma, T_{1/2}, \sigma_+, str, \tau)$ such that for all assemblies with a seed located at the start, a tile of type t_m appears at coordinate $(x, \lfloor \frac{N}{2} \rfloor)$ with no other tiles to its left, and t_m does not appear anywhere else.*

Proof sketch. Let R be an explicitly bounded rectangle and $N = 2^n + k$ with $k < 2^n$ be the height of R. Without loss of generality, we assume that the specially marked horizontal sides of R being "Start" at the bottom and the other, "Finish", at the top. See Fig. 3 for an overview.

We use the IBC, DBC and U-Turn Systems to define our Middle Finding SFTAM TAS that finds $\frac{N}{2} = 2^{(n-1)} + \frac{k}{2}$. There are two copies of the IBC, named $IBC1$ and $IBC2$. We list the steps for the Middle Finding System, and omit the details due to space constraints.
0. Growing a column of support tiles until "Finish".

1. Using the IBC System $IBC1$ to find the height $N = 2^n + k$ of R. The support tiles from the previous step are its seed.
2. Returning to row number 2^n using the second IBC system $IBC2$, which then outputs the value of k
3. Copying k until 2^{n-1} (the middle of 2^n).
4. Halving k by eliminating its least significant bit.
5. Shifting $\frac{k}{2}$ to the left by a U-Turn System.
6. Going up by $\frac{k}{2}$ using the DBC system. ◀

Figure 3 The steps of the Middle Finding System process, starting from the seed that is in red on the right. The tile t_m (the red tile on the left) appears in the middle of two rows of tiles that are shown by start and finish.

4 Distinguishing order-1 cuboids by their genus

Our main result is stated as follows.

▶ **Theorem 12** (Main Theorem). *There is a SFTAM tile self-assembly system $\mathcal{S}_\mathcal{G} = (\Sigma, T, \sigma, str, \tau)$ and a subset of tile-types $Y = \{t_{reg}, t_{mfs}\} \cup T_{ibc} \subseteq T$ such that for any order-1 cuboid $C = C_0 \setminus C_0'$ with the dimensions at least 10 for C_0', if $\mathcal{S}_\mathcal{G}$ assembles on C starting from a seed which is placed in a normal placement, the following holds:*

- *If C has genus 1, every terminal assembly of $\mathcal{S}$ on C contains at least one tile of Y.*
- *If C has genus 0, then no tile of Y appears in any producible assembly of $\mathcal{S}$ on C.*

The system presented here works for the case where the seed is on a *normal placement*, i.e. "far enough" from the borders of the surface.

▶ **Definition 13** (Normal placement). *Let C be an order-1 cuboid such that (x_1, y_1, z_1), (x_2, y_2, z_2), ... ,(x_n, y_n, z_n) are its vertices. A placement $p \in Pl(C)$ with position (x, y, z) is a* normal placement *of C if and only if for all $i \in \mathbb{N}$, two of the following inequalities hold:*

$\mid x_i - x \mid \geq 3\log(N) + 6$, $\mid y_i - y \mid \geq 3\log(N) + 6$ *and* $\mid z_i - z \mid \geq 3\log(N) + 6$, *where* N *is the largest of the three dimensions of the cuboid. The set of all normal placements is denoted by* $Pl_N(C)$.

The simplest example to demonstrate the concept of normal placement is on an order-0 cuboid $C \in O_0$. In this case, normal placements consist of the cuboid's surface minus its "frame" i.e. the border of the cuboid's edges with a thick margin. Hence there are 6 disconnected areas on C's faces where the normal placements are. The normal placements on order-1 cuboids can be described similarly. It should be noted that in this case there can be more than 6 disconnected areas. Note that in order to have normal placements at all, a cuboid needs to be large enough. Also, when dimensions of the order-1 cuboid are large enough and not too disproportionate, most placements are normal placements.

4.1 Region partition on order-1 cuboids

Let $C = C_0 \setminus C_0'$ be an order-1 cuboid. In order to detect a potential tunnel whose entrances are on parallel faces, the construction separates these faces. For this purpose we use three planes, one for each pair of parallel faces of C, located between them. Let P_X, P_Y, P_Z be three planes in this way: take $p \in Pl_N(C)$. The plane P_X is passing on p and is parallel to the plane formed by the Y-axis and Z-axis. The plane P_Y is parallel to the plane formed by the X-axis and Z-axis and is passing on p. The plane P_Z is parallel to the plane formed by the X-axis and Y-axis and contains the center of C_0. In Fig. 4 the seed in yellow is in the point p and the plane P_X, P_Y and P_Z are framed respectively by the ribbons R_X (in red), R_Y (in green) and R_Z (in blue) on C. For $i \in \{X, Y\}$, R_i is the connected component of $\partial C \cap P_i$ that contains p. If R_X and R_Y intersect in one point, R_Z in the empty set. If they intersect in two points, $R'_Z = P_Z \cap \partial C$ and R_Z is the connected component of R'_Z that has an intersection with R_Y. The difference $C \setminus \{R_X, R_Y, R_Z\}$ consists of up to 8 connected components called *regions*. They are noted by R_{XYZ} such that $X, Y, Z \in \{0, 1\}$ where 0 represents the left, down and back sides, and 1 represents the right, up and front sides. For example, R_{101} refers to the region at the right, down and front side of C. The parity of the regions R_{XYZ} is the parity of $X + Y + Z$. This way of partitioning C helps to define the graph G_C, the *region graph* of C:

▶ **Definition 14** (Region graph). *Let* $C = C_0 \setminus C_0^{'}$ *be an order-1 cuboid with* p *as a position of it, the planes* P_X, P_Y *two perpendicular planes passing on* p, *and* P_Z *a plane perpendicular to both planes passing through the middle of* P_Y. *Also, let* $R_i = \partial C_0 \cap P_i$ *for* $i \in \{X, Y, Z\}$. *There is a graph assigned to* C *named the* region graph $G_C(p)$ *whose vertices are the regions separated by* R_X, R_Y *and an edge is added between two regions if and only if they share* P_X, P_Y *or* P_Z.

For an order-0 cuboid C, $G_C(p)$ is a bipartite graph and therefore it is 2-colorable. The region graph for an order-0 cuboid is presented in Fig. 7.

If C is an order-1 cuboid with a tunnel, the number of disconnected regions can be less than 8 depending on the intersection of the tunnel with the three planes P_X, P_Y and P_Z. The *axis* of the tunnel is the direction orthogonal to its entrances. The three planes can intersect the tunnel in two ways: along the width of the tunnel when the plane is perpendicular to the axis of the tunnel, or along the length of the tunnel when the plane is parallel to the axis of the tunnel. Thus, a tunnel may have an intersection with up to three perpendicular planes, one along the width, and up to two other planes along the length. Based on this, three types of partitions into regions are possible and the possible numbers of regions are: 7

regions when one plane intersects along the width of the tunnel, 5 regions when one plane intersects along the length of the tunnel and one along the width (See Fig 11), and 1 region when the three perpendicular planes intersect along the tunnel, one along the width and the others along the length.

4.2 Overview of the assemblies of the genus detector $\mathcal{S}_\mathcal{G}$ on O_1

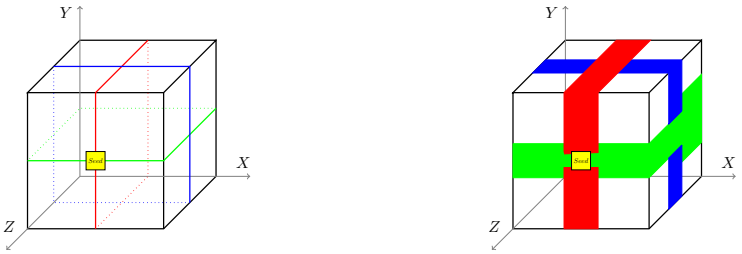


Figure 4 The skeleton of a terminal assembly of $\mathcal{S}_\mathcal{G}$ on an order-0 cuboid starting from a seed (in yellow) in a normal placement. On the left, the traces of the ribbons R_X (in red), R_Y (in green) and R_Z (in blue). On the right, the shape of the skeleton on the cuboid.

Let C be an order-1 cuboid. An assembly of $\mathcal{S}_\mathcal{G}$ starts from a seed in an arbitrary normal placement on C. In the TAS $\mathcal{S}_\mathcal{G}$, the seed acts like a compass for the assemblies. Without loss of generality, we assume that the side on which the seed is located is the face parallel to the XY-plane and intersects the Z axis, and the north label of the seed's tile points towards the Y axis. The process of the assemblies' growth in S_G consists of two phases, a phase for forming a *skeleton*, and a phase for filling up the skeleton:

1. Constructing the skeleton of the assembly's structures by at most 7 perpendicular ribbons on C. Here, the planes P_X, P_Y and P_Z are located from being framed by several ribbons of tiles (R_X, R_Y and R_Z) during the assembly and each step starts only when the previous step is finished.
   - R_X including one ribbon for framing the first plane P_X
   - R_Y including two ribbons for framing the second plane P_Y
   - R_Z including zero or four ribbons constitutes the frame of the third plane P_Z (depending on the intersection of the two previous planes, details will be given later)
2. Filling the inside of the assembly's skeleton by distinctive tiles. In this step the interior of the regions is partially filled by their distinctive tiles in a way that no connected component has a neighbor with the same inner filling tile.

In order to simplify the explanation of the process of the assemblies, first phase one is presented:

- how the skeleton grows depends on the placement of the seed
- how the skeleton partitions C into distinct connected components
- what its assigned region graph is.

Next, we study the phase of inner filling. Afterwards, we conclude the proof of the main theorem.

4.2.1 Terminal assemblies on order-0 cuboids

This section will characterise the set $A_\square^{C_0}[\mathcal{S}_\mathcal{G}]$ of terminal assemblies on an order-0 cuboid C_0. In fact, for the study of the shape of the productions in O_1^t, the productions on O_0 will be useful as a reference. We show that $\mathcal{S}_\mathcal{G}$ partitions order-0 cuboids into eight distinct regions as presented in Section 4.1. Later in the next section we study the case of order-1 cuboids.

1. The structure of the skeleton.

▶ **Lemma 15.** *Let $C \in O_0$ be an order-0 cuboid and assume that the seed σ is placed at a normal placement $p \in Pl_N(C)$. Every terminal assembly of $\mathcal{S}_\mathcal{G}$ on C includes a "3-step skeleton" noted by $R_X \cup R_Y \cup R_Z$ where each part is located on the corresponding ribbon defined in Section 4.1.*

Proof. Every terminal assembly of $\mathcal{S}_\mathcal{G}$ on C includes a "3-step skeleton" denoted by $R_X \cup R_Y \cup R_Z$ where each part is located on the corresponding ribbon defined in Section 4.1. In the first step, tiles make a vertical segment ribbon of tiles around C to form R_X, starting from the south of the seed and finishing at its north. R_X divides C into two regions, the *right side* and the *left side* of C with respect to the seed σ. (We always assume that we view the cuboid from the point of view of the Z-axis, as in Fig. 4, and thus *left*, *right*, *up*, *down*, *back*, *front* refer to this point of view.) Next, R_Y starts to form only when R_X rebounds. R_Y consist of two segment ribbons starting from both the right and left sides of σ. They develop perpendicular to R_X by using two Middle Finding Systems (Definition 11), one on each side. Note that each system has its own distinguished tile types. Once the R_Y ribbons form, they separate C into an *up side* and a *down side*. Thus, C is now partitioned into four separate regions due to the first and second step ribbons. Next, by finding the middle of each of R_Y's ribbons on the right and left faces, four new perpendicular ribbons are generated at the right-up and left-down sides from the tile of type t_m in the Middle Finding Systems. They go on, until they reach R_X on the upper and down faces. The plane R_Z passes through the union of these four ribbons. This step creates a separation between the *front side* and the *back side* of the cuboid C with respect to σ. We show the detailed assembly of R_X, R_Y and R_Z in Fig. 6 that starts from the seed σ (in red) at the left of the figure, where R_X rebounds on C. The western two-side Middle Finding System and its assign parts of R_Z is omitted for the sake of brevity, however they are the mirror image of the eastern ones. ◀

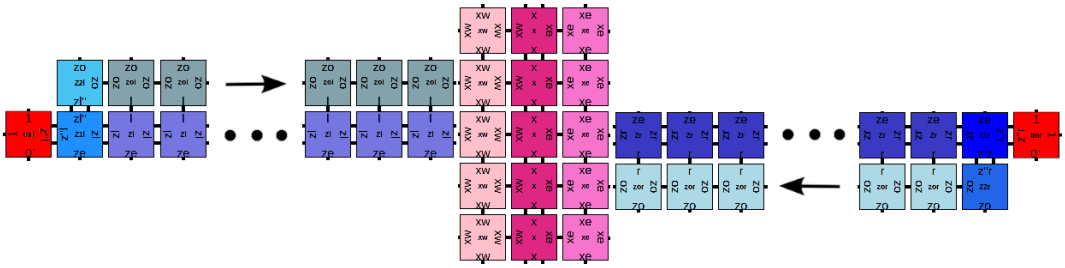


Figure 5 Two ribbons of R_Z meeting the ribbon R_X.

As mentioned in Section 4.1, these three steps partition C into 8 distinct regions.

2. Inner filling of the skeleton. After the formation of the skeleton, the second phase is to fill the eight regions by lines of interior tiles, once the R_Z ribbons reach R_X. Since the region graph of an order-0 cuboid C is a 2-colorable graph, we use two tile types to distinctly tile C.

▶ **Lemma 16.** *Let $C \in O_0$ be an order-0 cuboid. For all the terminal assemblies $\alpha \in A^{C_0}_{\square}[\mathcal{S}_\mathcal{G}]$ started from a seed $\sigma \in Pl_N(C)$, the tile $t_{even} = (z_e, x_{ev}, z_e, x_{ev})$ appears in the even regions and the tile $t_{odd} = (z_o, x_{od}, z_o, x_{od})$ appears in the odd regions.*

Proof. First, four ribbons of tiles types (see Fig. 8) appear at the intersection of P_X and the P_Z ribbons. Then, from the ribbons along R_X, straight lines of tiles start growing parallel to the x axis using strength 2 glues x_{ev} (resp. x_{od}) in even (resp. odd) regions. Thanks to

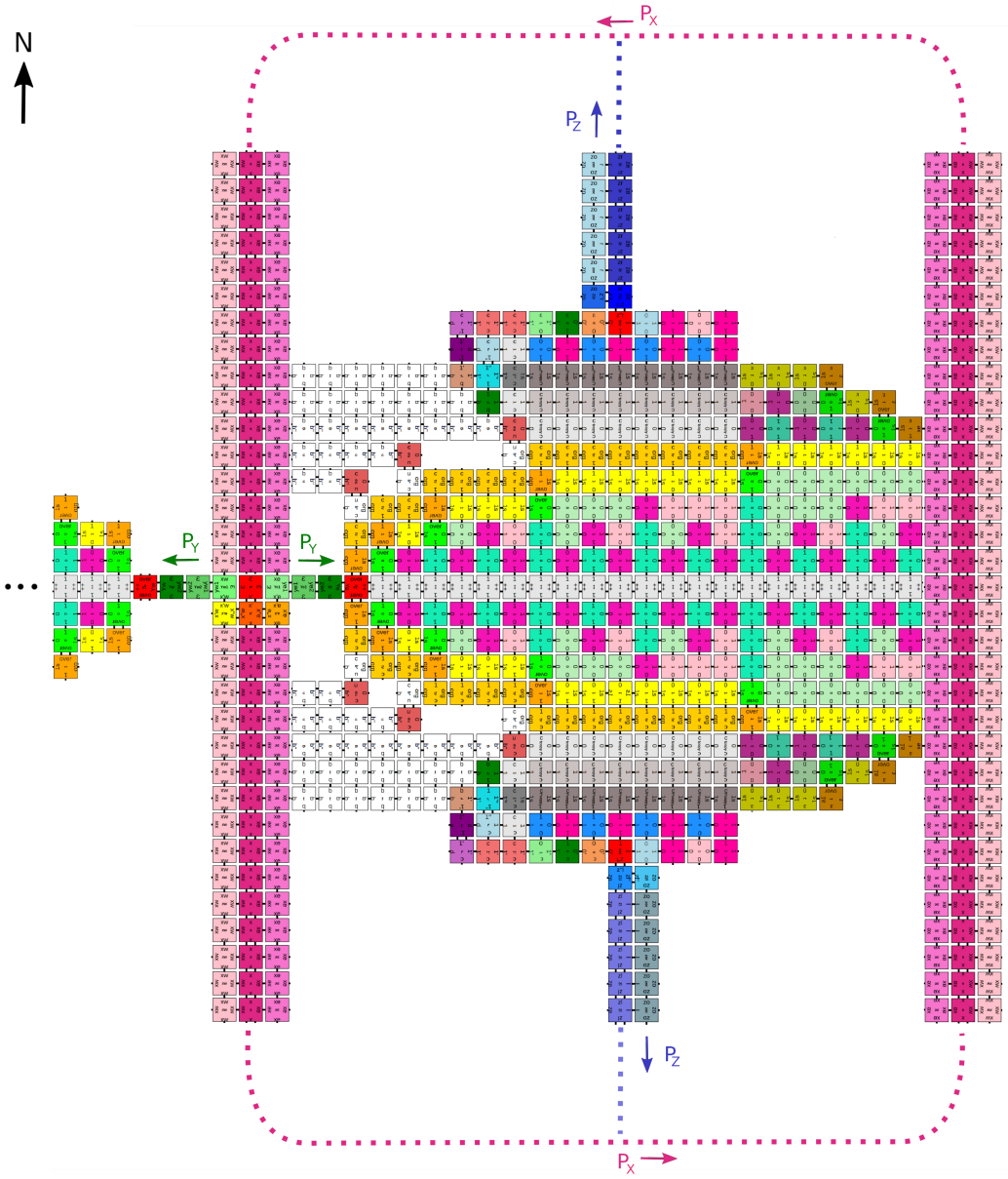
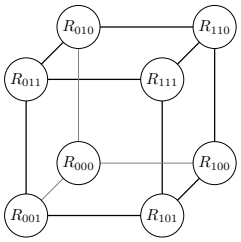
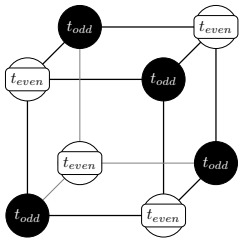


Figure 6 The assembly of R_X, R_Y and R_Z on an order-0 cuboid. The seed is located in the middle of R_X at the left. R_X grows from the south of the seed and finishes at its north. Then, P_Y starts growing by two two-side eastern and western Middle Finding Systems. At the end, P_Z starts to assemble from the found middle tile of R_Y (in red) and finishes by arriving at P_X. The western two-side Middle Finding System and its assigned parts of R_Z are omitted for the sake of brevity, however they are the mirror image of the eastern ones.

modulo 5 counters on the even R_X border tiles, there is one such line every other 5 position along that part of the border with tiles of type $t_{even} = (z_e, x_{ev}, z_e, x_{ev})$. Also, on the odd R_X border tiles, thanks to modulo 3 counters on the odd R_X border tiles, there is one such line every other 3 position along that part of the border with tiles of type $t_{odd} = (z_o, x_{od}, z_o, x_{od})$. These lines form the even (resp. odd) filling tiles and fill the partitioned regions. See Fig. 10 for an illustration.



(a) The region graph of an order-0 cuboid.



(b) Tile coloring of the region graph, and filling up the regions via tiles of types t_{even} and t_{odd}.

Figure 7 The region graph G_C and its 2-coloring.

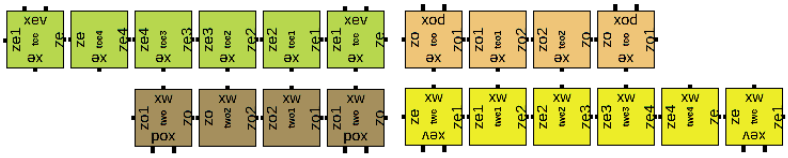


Figure 8 The tile types of four inner ribbons at the intersection of R_X and R_Z.

The 2-coloring indicates also where the tiles of type t_{even} and of type t_{odd} can be placed. The region R_{000} and the regions with even distance to it are tiled by tiles of type t_{even}, and the ones with odd distance to it, by t_{odd} tiles. For more clarity, see Fig. 7 where the regions corresponding to t_{even} are colored white and the ones with t_{odd} are colored black. ◀

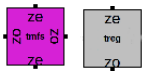


Figure 9 The tile types t_{mfs} and t_{reg} (which may only appear if C has genus 1).

4.2.2 Terminal assemblies on order-1 cuboids with genus 1

We consider now the process of the assembly of $\mathcal{S}_\mathcal{G}$ for order-1 cuboids with a tunnel. This section will characterise the set $A^{C^t_1}_{\square}[\mathcal{S}_\mathcal{G}]$ of terminal assemblies on such order-1 cuboids.

Let C be an order-1 cuboid with a tunnel. The key element of the proof is the appearance of some specific tile in each assembly since it has less than 8 regions. The assemblies on C have a skeleton with a different shape depending on the region graph associated with the placement of the seed. Let P_i and R_i for $i \in \{X, Y, Z\}$ be defined as presented in Section 4.1. If a plane P_i intersects along the width of the tunnel, it acts like a separator between the two parallel faces where the tunnel's entrances are located. If a plane P_i intersects along the length of the tunnel, the tiles of R_i enter and pass inside the tunnel. Moreover, three types of partitions into regions are possible and the possible numbers of regions are: 7 regions when one plane intersects along the width of the tunnel, 5 regions when one plane intersects along the length of the tunnel and one along the width, and 1 region when three perpendicular planes intersect along the tunnel, one along the width and the others along the length. Each case needs to be studied separately, we give the proof of the case with 5 regions and the proof of the Lemma 17 and Lemma 19 exist in the appendix.

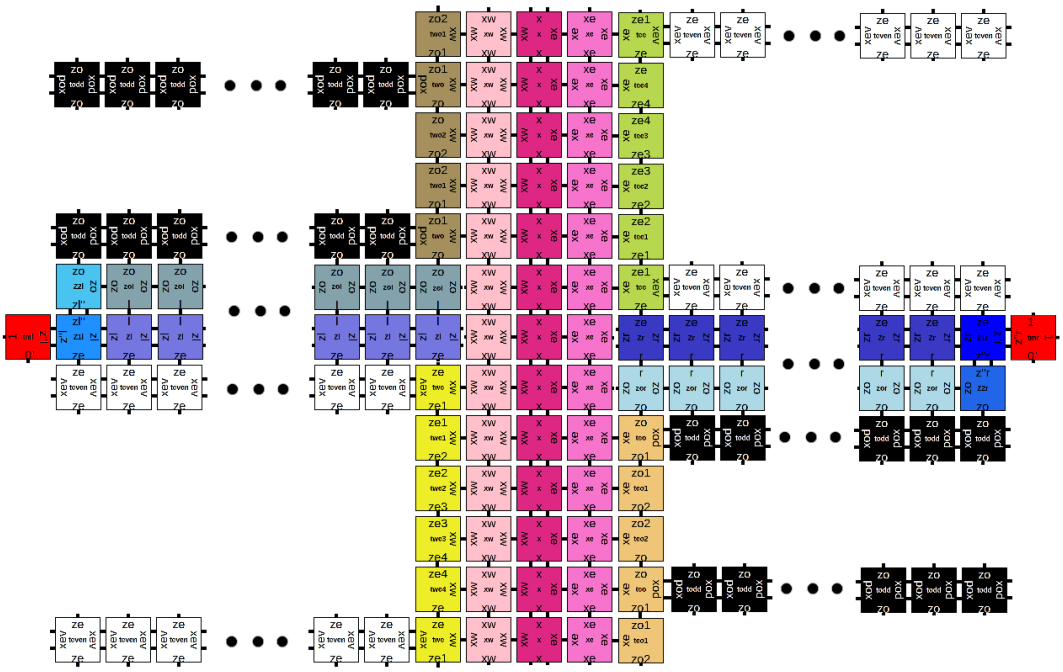


Figure 10 The inner filling with tiles of types t_{even} (white) and t_{odd} (black) at the two places where the R_Z ribbons meet R_X.

Note that in what follows, $G_C(\sigma)$ refers to the region graph $G_C(p)$ such that p is the position of the seed σ on C.

Case 1 (7 regions): one plane intersects along the width of the tunnel. In this case, tiles of types t_{odd} and t_{even} touch, which enforces the attachment of t_{reg} or t_{mfs}.

▶ **Lemma 17.** *Let $C = C_0 \setminus C_0' \in O_1^t$ be an order-1 cuboid with the dimensions at least 10 for C_0'. Assume that the seed σ is placed in a normal placement $p \in Pl(C)$. In a terminal assembly of the system $S_\mathcal{G}$, if only one of the planes defined in Section 4.1 intersect with the tunnel, $G_C(\sigma)$ has 7 regions and a tile of type t_{reg} or t_{mfs} appears in the assembly.*

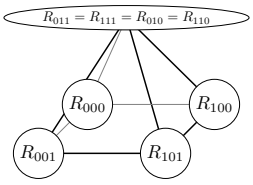
Case 2 (5 regions): the tunnel intersects with P_Z, and exactly one of P_X and P_Y.

▶ **Lemma 18.** *Let $C \in O_1^t$ be an order-1 cuboid and assume that the seed σ is placed in a normal placement $p \in Pl(C)$. In a terminal assembly of the system $S_\mathcal{G}$, if the plane P_Z and exactly one of the planes P_X and P_Y defined in Section 4.1 have an intersection with the tunnel, there exist 5 regions on the cuboid and a tile of type t_{mfs} appears in the assembly.*

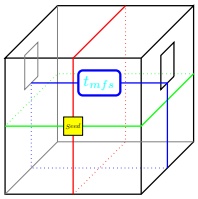
Proof. If the seed is placed where the tunnel has intersection with two perpendicular planes, one of them intersects the tunnel along its width and the other one along its length. If P_Z intersects with the tunnel along the length, the ribbons of R_Z meet each other inside the tunnel. However, if P_Z intersects the tunnel along its width, they meet outside the tunnel.

In both cases, the tile $t_{mfs} = (z_e, z_o, z_e, z_o)$ appears in the assembly when two frame ribbons of P_Z meet each other. Note that when the tunnel has intersection with P_Z and one of the planes P_X or P_Y, the cuboid is separated into two connected components such that one of them is a cuboid with genus 0 and the other one is a cuboid with genus 1. The

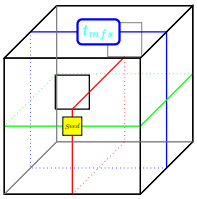
part with genus 0 has 4 distinct regions, and the part with genus 1 (containing a tile of type t_{mfs}) has one single region. In total, there exist 5 distinct regions on the cuboid C. For an illustration of the skeleton and its graph in this case, see Fig. 11. ◀



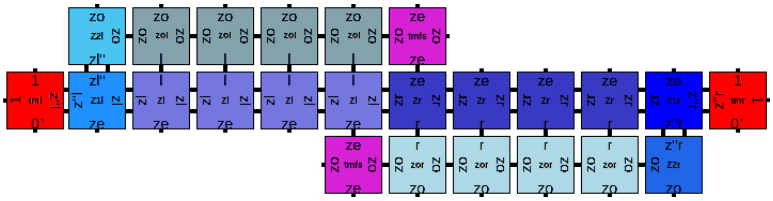
(a) The region graph.



(b) The tunnel intersects along the width of plane P_X and length of plane P_Z.



(c) The tunnel intersects along the length of plane P_X and width of plane P_Z.



(d) A tile of type t_{mfs} appears if and only if two segments of P_Z (in purple) intersect each other by passing through a tunnel (instead of reaching P_X).

Figure 11 The case where C is partitioned into 5 distinct regions.

Case 3 (1 region): the tunnel intersects with P_X and P_Y. Here, two opposite ribbons of R_Y meet instead of colliding with R_X. This enforces the attachment of some tiles in $T_{ibc} \subseteq Y$.

▶ **Lemma 19.** *Let $C \in O_1^t$ be an order-1 cuboid and assume that the seed σ is placed in a normal placement $p \in Pl(C)$. In a terminal assembly of the system $S_{\mathcal{G}}$, if two planes P_X and P_Y defined in Section 4.1 intersect the tunnel, there exists 1 regions on the cuboid and a tile of one of the T_{ibc} types appears in the assembly.*

Note that the situation when the seed is located inside the tunnel is similar to Case 3, up to topological isomorphism.

From Lemmas 17, 18 and 19, the following corollary is obtained:

▶ **Corollary 20.** *Let $C = C_0 \setminus C_0' \in O_1$ be an order-1 cuboid with the dimensions at least 10 for C_0' and α be an assembly of the TAS $\mathcal{S}_{\mathcal{G}} = (\Sigma, T, \sigma, str, \tau)$ such that its seed is placed at a normal placement. If there is tunnel on C (i.e. its genus is 1), at least a tile type from $Y = \{t_{reg}\} \cup \{t_{mfs}\} \cup T_{ibc} \subseteq T$ exists in all terminal assemblies of $\mathcal{S}_{\mathcal{G}}$ on C.*

4.3 Detecting the genus of the order-1 cuboids via $\mathcal{S}_{\mathcal{G}}$

Before proving our main theorem (Theorem 12), we need to prove the following lemma:

▶ **Lemma 21.** *Let C be an order-1 cuboid. If one tile of $Y=\{t_{reg}\} \cup \{t_{mfs}\} \cup T_{ibc} \subseteq T$ exists in a terminal assembly of $\mathcal{S}_{\mathcal{G}}$ on C starting from a seed in a normal placement, there is a tunnel on C.*

Proof. If a tile of type t_{mfs} exists in a terminal assembly on C, two cases are possible. In one case, there is a tunnel that intersects only P_X along its width and t_{even} and t_{odd} intersect each other perpendicularly. As a result, t_{mfs} appears in the assembly. In the other case, two ribbons of R_Z must meet each other since the tiles whose labels correspond to the labels of t_{mfs} are those of the R_Z ribbons. Recall that the R_X and R_Y ribbons intersect at two places : one at the seed (since R_Y grows out of R_X) and a second time, where the tiles of type t_{eu}, t_{ed}, t_{wu} or t_{wd} appear in the assembly as the row tile number 1, in the second IBC system of the Middle Finding Systems. The two ribbons of R_Z, together with the parts of the Middle Finding System located between the second intersection of R_X and R_Y on the one hand, and R_Z on the other hand, form a closed ribbon on the surface of C (highlighted in green and blue Fig. 12). This ribbon and R_X pass through each other perpendicularly at only one place. Since they pass through each other perpendicularly, it can be concluded that the cuboid C cannot be topologically homeomorphic to the sphere, or in other words, be a genus 0 cuboid, and so a tunnel must exist. The cases where there is a tile of type t_{reg} or $t \in T_{ibc}$ are similar. ◀

In the cases where the genus is 0 but there is a pit or concavity the construction also yields an 8 region partition but the details are omitted due to space constraints.

We are now ready to prove Theorem 12.

Proof of Theorem 12. Let $C = C_0 \setminus C_0' \in O_1$ be an order-1 cuboid with the dimensions at least 10 for C_0' and α be an assembly of the TAS $\mathcal{S}_\mathcal{G} = (\Sigma, T, \sigma, str, \tau)$ such that its seed is placed at a normal placement. Note that if C_0 is too small, there is no normal placement. By Corollary 20 and Lemma 21, there is a tile type from $Y = \{t_{reg}\} \cup \{t_{mfs}\} \cup T_{ibc} \subseteq T$ in all terminal assemblies of $\mathcal{S}_\mathcal{G}$ on C if and only if there is tunnel on C (i.e. its genus is 1). ◀

Figure 12 The closed ribbon formed by parts of the Middle Finding System (green) and the two ribbons of R_Z (blue), when two R_Z ribbons meet each other instead of reaching R_X. They meet the red ribbon R_X only once.

The general principle of the construction is as follows: cut the order-1 cuboid into regions and check if the partition is the same as it would be on a cube. If it is the case, the cuboid has genus 0, eight regions and the tiles of Y cannot be used in any terminal assembly. Otherwise, the cuboid has genus 1, less than eight regions, and the tiles of Y must be used in any terminal assembly.

In fact, the SFTAM system we obtain works quite intuitively. The different "moving parts" are necessary to ensure the good function of the system:

- Firstly, the Middle Finding System ensures the shape is split along each dimension. In fact, a precise control is necessary to prevent false positives, i.e. order-1 cuboids of genus 0 with a tile of Y in their assembly, and less than 8 regions. To do so, the partition ensures that any tunnel lies between two different regions that have the possibility of sharing a tunnel. Hence, the Middle Finding System is needed to avoid false positives.
- Lastly, the filling with unequally spaced stripes ensures that there is enough empty space which triggers the detection of the meeting of two regions.

The relative complexity of these illustrates the challenges of working on an unknown surface.

5 Conclusion

We have introduced our new model, SFTAM, that enables to perform tile self-assembly on 3D surfaces. We have shown that we can use it to determine the genus of a given surface. For this, we have worked on a simple and special family of polycubes, the order-1 cuboids.

It would be interesting to extend our results to a larger family of polycubes. In our work, the Middle Finding System was used to detect a potential tunnel on an order-1 cuboid. However, for more complicated surfaces, one needs to ensure that some part of the construction does go through the tunnel, and that it can be differentiated from the tiles it meets on the other side. The idea of having regions with distinct identities can be reused in this context, but the Middle Finding System needs to be supplemented or replaced.

References

1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik D Demaine, Martin L Demaine, Robin Flatland, Scott D Kominers, and Robert Schweller. Shape replication through self-assembly and rnase enzymes. In *Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms*, pages 1045–1064. SIAM, 2010.

2 Greg Aloupis, Prosenjit K Bose, Sébastien Collette, Erik D Demaine, Martin L Demaine, Karim Douieb, Vida Dujmović, John Iacono, Stefan Langerman, and Pat Morin. Common unfoldings of polyominoes and polycubes. In *Computational Geometry, Graphs and Applications: 9th International Conference, CGGA 2010, Dalian, China, November 3-6, 2010, Revised Selected Papers*, pages 44–54. Springer, 2011.

3 Robert D Barish, Rebecca Schulman, Paul WK Rothemund, and Erik Winfree. An information-bearing seed for nucleating algorithmic self-assembly. *Proceedings of the National Academy of Sciences*, 106(15):6054–6059, 2009.

4 Matthew Cook, Tristan Stérin, and Damien Woods. Small Tile Sets That Compute While Solving Mazes. In *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DNA.27.8.

5 Jérôme Durand-Lose, Jacob Hendricks, Matthew J Patitz, Ian Perkins, and Michael Sharp. Self-assembly of 3-d structures using 2-d folding tiles. *Natural Computing*, 19:337–355, 2020.

6 Matthew R Jones, Nadrian C Seeman, and Chad A Mirkin. Programmable materials and the nature of the dna bond. *Science*, 347(6224):1260901, 2015.

7 Wenyan Liu, Hong Zhong, Risheng Wang, and Nadrian C Seeman. Crystalline two-dimensional dna-origami arrays. *Angewandte Chemie International Edition*, 50(1):264–267, 2011.

8 Kao Ming-Yang and Vijay Ramachandran. Dna self-assembly for constructing 3d boxes. In *International Symposium on Algorithms and Computation*, pages 429–441. Springer, 2001.

9 Matthew J Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13:195–224, 2014.

10 Paul WK Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

11 Paul WK Rothemund and Erik Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468, 2000.

12 Ned et al." "Seeman. Dna nanotechnology: Bibliography from ned seeman's laboratory, 2023. URL: http://seemanlab4.chem.nyu.edu/nanobib.html.

13 Vyankat A Sontakke and Yohei Yokobayashi. Programmable macroscopic self-assembly of dna-decorated hydrogels. *Journal of the American Chemical Society*, 144(5):2149–2155, 2022.

14 Hao Wang. Proving theorems by pattern recognition—ii. *Bell system technical journal*, 40(1):1–41, 1961.

15 Erik Winfree. *Algorithmic self-assembly of DNA*. California Institute of Technology, 1998.

16 Joseph F Woods, Lucía Gallego, Pauline Pfister, Mounir Maaloum, Andreas Vargas Jentzsch, and Michel Rickhaus. Shape-assisted self-assembly. *Nature Communications*, 13(1):3681, 2022.
17 Shuguang Zhang. Fabrication of novel biomaterials through molecular self-assembly. *Nature biotechnology*, 21(10):1171–1178, 2003.
18 Rebecca Zhuo, Feng Zhou, Xiaojin He, Ruojie Sha, Nadrian C Seeman, and Paul M Chaikin. Litters of self-replicating origami cross-tiles. *Proceedings of the National Academy of Sciences*, 116(6):1952–1957, 2019.

A Technical Details

In our assemblies, we will represent a number by tiles, in a classic way for tile self-assembly (see e.g. [11]) as follows.

▶ **Definition 22** (Row tile number). *Let T_0 and T_1 be two sets of tiles with labels 0 and 1, respectively. Let N be an integer and $a_1...a_2a_n$ be its binary representation with $n = \lceil log_2 N \rceil$. A row of tiles with labels $a_1^*, a_2, ..., a_n$ is the* row tile number *representation of N such that the distinct tile a_1^*, represents the most significant bit of the number. See Fig. 13 for an example.*

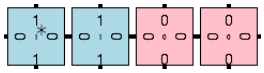
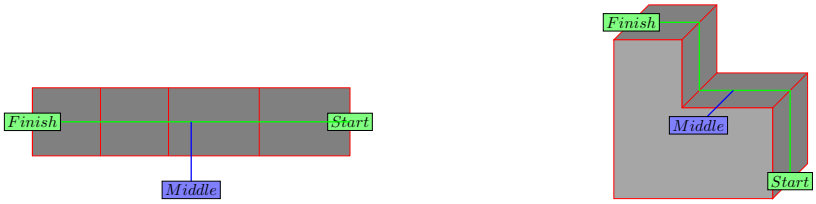


Figure 13 Representation of the number 12 by its row tile number.



(a) On $\mathbb{Z}^2$. **(b)** On a polycube.

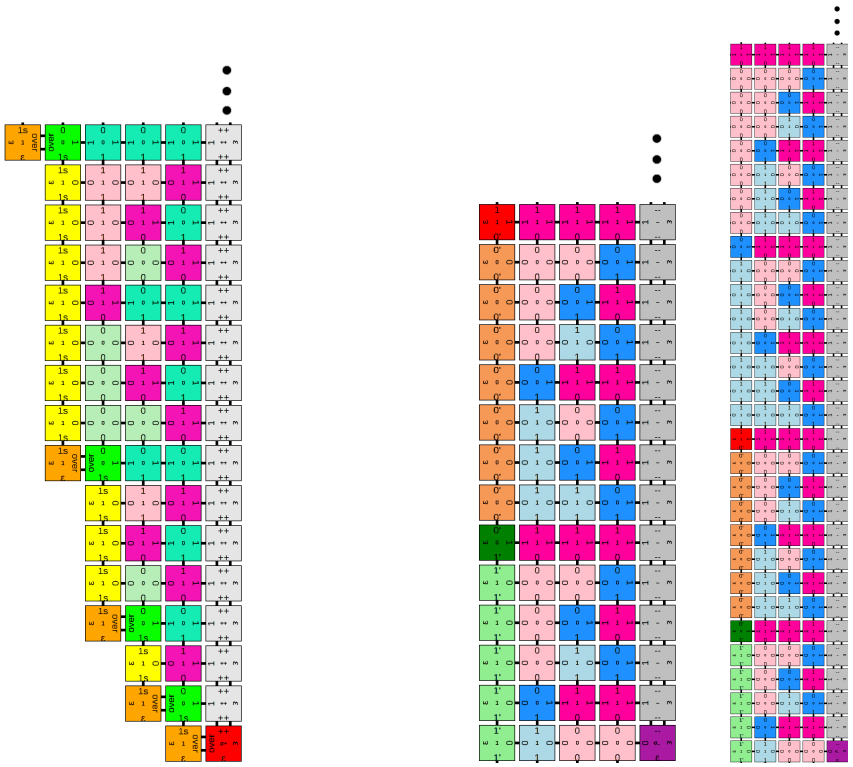
Figure 14 Finding the middle of a surface. The underlying rectangle is in dark gray.

B The order-1 cuboids with pit or concavity

In the case that $C \in O_1$ is an order-1 cuboid with a concavity or a pit (whose genus is 0), the assembly's process is similar to the assembly on order-0 cuboids. The frame ribbons form completely by the assumption that the seed is located on a normal placement of C, and separate C into 8 distinct regions, and the insides of the regions are tiled independently by inner tiles of types t_{odd} and t_{even}. However, in the case of a concavity, the regions do not necessarily meet edge to edge, see Fig. 17 for an illustration.

C Omitted proofs

Proof of Lemma 17. Let the seed be placed in a manner that only one of the planes P_X, P_Y or P_Z intersects along the width of the tunnel.



(a) The assembly of the IBC System until the number 16.

(b) The assembly of the DBC System for number 12 to −1 at the left and negative 15 at the right.

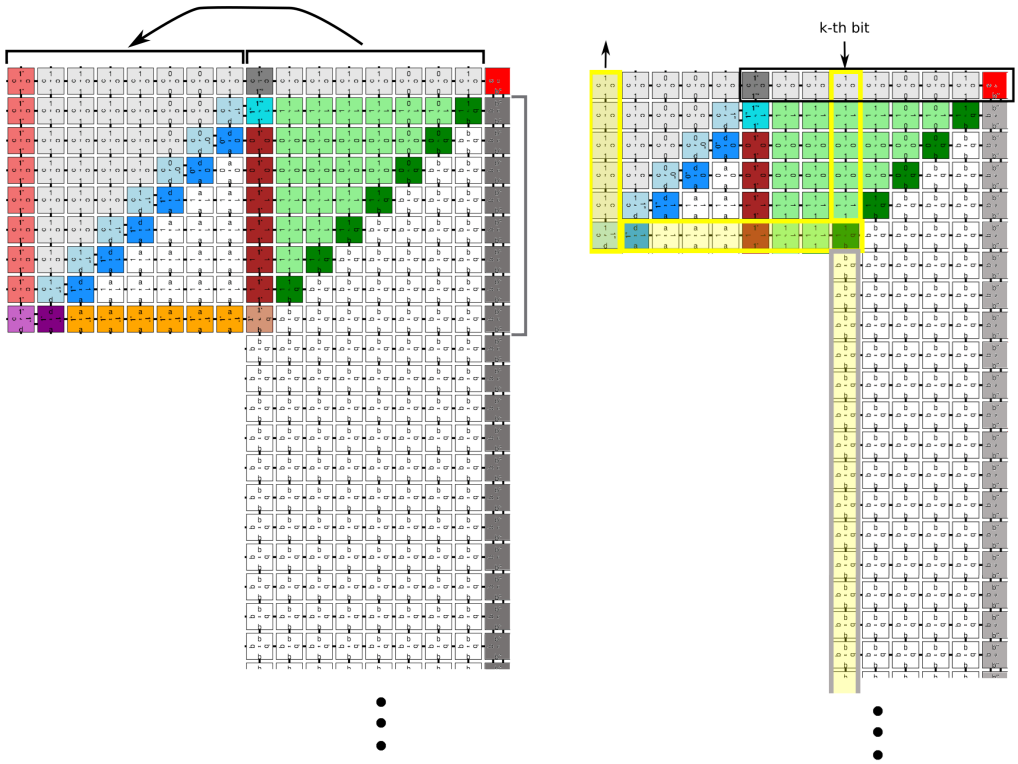
Figure 15 Increasing Binary Counter System (left) and Decreasing Binary Counter System (right).

The plane that intersects the tunnel is the separating buffer of two regions R_{xyz} and $R_{x'y'z'}$ containing the two tunnel's entrances. In this case, the two regions R_{xyz} and $R_{x'y'z'}$ are joined by the tunnel and get combined into a single region via the tunnel. Therefore, the number of distinct regions decreases to 7 regions (compared with the genus 0 case, where we always have 8 regions). See Fig. 18 for an illustration.

Without loss of generality, assume that $x + y + z$ is an odd number and $x' + y' + z'$ is an even number. When two regions R_{xyz} and $R_{x'y'z'}$ are joined by the tunnel, tiles of type $t_{odd} = (z_o, x_{od}, z_o, x_{od})$ from R_{xyz} and of type $t_{even} = (z_e, x_{ev}, z_e, x_{ev})$ from $R_{x'y'z'}$ both exist in the new unique region. We show that the tile type $t_{reg} = (z_e, x_{od}, z_o, x_{ev})$ or t_{mfs} must then occur in the assembly. The tile types t_{reg} and t_{mfs} are the only tile type of $\mathcal{S}_\mathcal{G}$ with labels z_o and z_e of inner filling tiles t_{odd} and t_{even}. To conclude the proof, one needs to show that in a region with a disconnected border, there is a *good* empty space, that is an empty space which sees both an even tile and an odd tile through strength 1 sides. Then, this space can be filled by neither type of filling tiles, but it must eventually be filled by a tile of type $t_{reg} = (z_e, x_{od}, z_o, x_{ev})$ or $t_{mfs} = (z_e, z_o, z_e, z_o)$. In a region with a tunnel, on each side of the tunnel, the border of every 10×10 square must be crossed by either

- at least two of the lines of tiles starting from P_X on that side of the tunnel, or
- at least two of the lines exiting the tunnel.

In particular, because C'_0 is at least 10×10 units wide, there are at least two lines crossing one of the edges the tunnel in the same direction. Each such line must either reach the



(a) Copying 11111001 in the U-Turn System. The gray bracket on the right shows the minimum number of support tiles that are necessary for this assembly.

(b) The k-th stage of the assembly in the U-Turn System is shown by yellow-filled rectangles. The value of the k-th significant bit is copied down by $k-1$ rows during the previous stages. The k-th stage copies the value one time to the south and n times to the left and finally k times to the top. Here, $k = 5$ and $n = 8$. In addition, in the k-th stage, the tiles of type $t^b_{\longleftarrow}$ in the gray rectangle appear below the tile of type $t_{\lrcorner}$, and they will be the supports for the $(k+1)$-th stage. The seed is highlighted in the black rectangle.

Figure 16 U-Turn System.

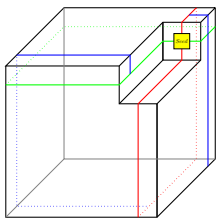


Figure 17 Cuboid with concavity: R_X (red), R_Y (green) and R_Z (blue). The latter consists of two semi-planes.

opposite connected component of the border, be stopped orthogonally by a line from the opposite side of the tunnel, or run head-first into an opposite line. Consider such a pair of lines, with minimal distance between them. In particular, that distance must be at most 10.

- If one of the lines reaches the opposite connected component of the border, either of the spaces next to its end is *good* and in this case the tile of type t_{reg} appears in the assembly;

- likewise, if one of them is stopped orthogonally by a line from the opposite side of the tunnel, one of the spaces next to the intersections is *good* and a tile of type t_{mfs} appears.

Moreover, if one of them runs head-first into an opposite line, the other cannot, because their distance cannot be at the same time divisible by 15, positive and less than 10. Hence the pair satisfies one the previous cases. This concludes the proof of that case of our construction. ◀

Proof of Lemma 19. In this case, the skeleton of the assembly is not the same as before. Recall the process of the assembly's skeleton: The frame ribbons of the plane P_X are generated independently from σ. Two segment ribbons of the plane P_Y begin to grow after rebounding on the plane P_X, regardless of passing through a tunnel or not. However, the ribbons of P_Z start to grow only after finding the middle of P_Y and they end by reaching the ribbon of P_X. Considering this process, when the two planes P_X and P_Y intersect with the tunnel, the plane P_Z is not able to form since there is a tunnel that does not permit to have the collision of P_Y and P_X, and the ribbons of P_Y do not end in P_X. As a result, since the plane P_Z depends on the collision of the P_Y ribbons with the P_X ribbons, P_Z is not able to form.

Moreover, two ribbons of P_Y must meet each other at a tile of one of the T_{ibc} types that comes between their $IBC1$ systems. This happens inside the tunnel if P_Y intersects the tunnel along its length, and outside the tunnel if it intersects the tunnel along its width. In either cases a tile of one of the T_{ibc} types appears.

Note that the skeleton consists of two closed loops of P_X ribbons and P_Y ribbons. This phenomenon demonstrates that the genus of C is 1. In order to have a better overview, see Fig. 19. Furthermore, all regions are united and there is only one single region throughout the whole surface of C. ◀

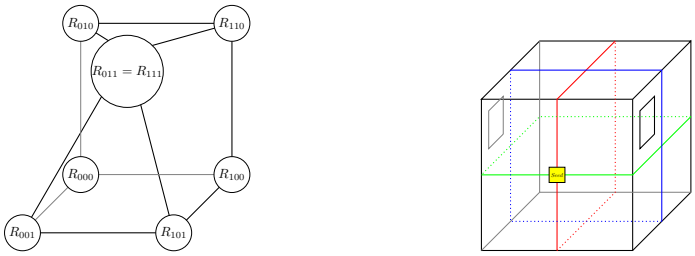
Proof of Corollary 20. If there is a tunnel on C, at least one of the planes P_X, P_Y and P_Z defined in Section 4.1 intersects with the tunnel since its entrances are on parallel faces of the cuboid, and these planes are located between parallel faces.

Firstly, if the tunnel of C intersects with only one of the planes, due to Lemma 17, a tile of type t_{reg} or t_{mfs}, which are the only tile types of $\mathcal{S}_\mathcal{G}$ with labels in common with both inner filling tile types t_{odd} and t_{even}, appears in the assembly.

Nextly, if two planes (among them P_Z) intersect with the tunnel on C, a tile of type t_{mfs} appears in all terminal assemblies on C by Lemma 18.

At the end, if two planes P_X and P_Y intersect with the tunnel, Lemma 19 implies that a tile of one of the T_{ibc} types is present in the assembly. ◀

The places where a tunnel implies the presence of a tile of Y are shown in Fig. 20.



(a) Region graph. **(b)** The cuboid and the three planes.

Figure 18 The case where $C \in O_1^t$ is partitioned into 7 distinct regions. If there is a tunnel between two distinct regions, a tile of type t_{reg} or t_{mfs}, which have common labels with both t_{even} and t_{odd}, must appear in the assembly.

(a) The tunnel intersects along the length of plane P_X and width of plane P_Y.

(b) The tunnel intersects along the width of plane P_X and length of plane P_Y.

Figure 19 Intersection of tunnel with the two planes P_X (red) and P_Y (green).

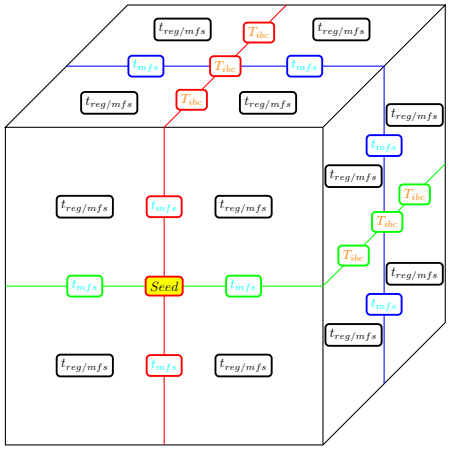


Figure 20 The places on a cuboid where, if there is a tunnel, a tile of Y must appear in the assembly.

On the Runtime of Chemical Reaction Networks Beyond Idealized Conditions

Anne Condon
University of British Columbia, Vancouver, Canada

Yuval Emek
Technion – Israel Institute of Technology, Haifa, Israel

Noga Harlev
Technion – Israel Institute of Technology, Haifa, Israel

Abstract

This paper studies the (discrete) *chemical reaction network (CRN)* computational model that emerged in the last two decades as an abstraction for molecular programming. The correctness of CRN protocols is typically established under one of two possible schedulers that determine how the execution advances: (1) a *stochastic scheduler* that obeys the (continuous time) Markov process dictated by the standard model of stochastic chemical kinetics; or (2) an *adversarial scheduler* whose only commitment is to maintain a certain fairness condition. The latter scheduler is justified by the fact that the former one crucially assumes "idealized conditions" that more often than not, do not hold in real wet-lab experiments. However, when it comes to analyzing the *runtime* of CRN protocols, the existing literature focuses strictly on the stochastic scheduler, thus raising the research question that drives this work: Is there a meaningful way to quantify the runtime of CRNs without the idealized conditions assumption?

The main conceptual contribution of the current paper is to answer this question in the affirmative, formulating a new runtime measure for CRN protocols that does not rely on idealized conditions. This runtime measure is based on an adapted (weaker) fairness condition as well as a novel scheme that enables partitioning the execution into short *rounds* and charging the runtime for each round individually (inspired by definitions for the runtime of asynchronous distributed algorithms). Following that, we turn to investigate various fundamental computational tasks and establish (often tight) bounds on the runtime of the corresponding CRN protocols operating under the adversarial scheduler. This includes an almost complete chart of the runtime complexity landscape of predicate decidability tasks.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases chemical reaction networks, adversarial runtime, weak fairness, predicate decidability

Digital Object Identifier 10.4230/LIPIcs.DNA.29.3

Related Version *Full Version*: `http://arxiv.org/abs/2307.00647` [13]

Funding *Anne Condon*: This work was supported by an NSERC Discovery grant.
Yuval Emek: This work was supported by VATAT Fund to the Technion Artificial Intelligence Hub (Tech.AI).

1 Introduction

Chemical reaction networks (CRNs) are used to describe the evolution of interacting molecules in a solution [20] and more specifically, the behavior of regulatory networks in the cell [7]. In the last two decades, CRNs have also emerged as a computational model for molecular programming [24, 14]. A protocol in this model is specified by a set of species and a set of reactions, which consume molecules of some species and produce molecules of others.

In a (discrete) CRN computation, inputs are represented as (integral) molecular counts of designated species in the initial system configuration; a sequence of reactions ensues, repeatedly transforming the configuration, until molecular counts of other designated species represent the output. The importance of CRNs as a model of computation is underscored by the wide number of closely related models, including population protocols [2, 4], Petri nets [23], and vector addition systems [21].[1]

The standard model of stochastic chemical kinetics [20], referred to hereafter as the *standard stochastic model*, dictates that the execution of a CRN protocol (operating under fixed environmental conditions) advances as a continuous time Markov process, where the rate of each reaction is determined by the molecular counts of its reactants as well as a reaction specific rate coefficient. This model crucially assumes that the system is "well-mixed", and so any pair of distinct molecules is equally likely to interact, and that the rate coefficients remain fixed.[2] Under the standard stochastic model, CRNs can simulate Turing machines if a small error probability is tolerated [14]. The correctness of some protocols, including Turing machine simulations, depends sensitively on the "idealized conditions" of fixed rate coefficients and a well-mixed system.

However, correctness of many other CRN protocols, such as those which stably compute predicates and functions [2, 4, 11, 18, 8, 10], is premised on quite different assumptions: correct output should be produced on *all* "fair executions" of the protocol, which means that the correctness of these protocols does not depend on idealized conditions. These protocols operate under a notion of fairness, adopted originally in [2], requiring that reachable configurations are not starved; in the current paper, we refer to this fairness notion as *strong fairness*. A celebrated result of Angluin et al. [2, 4] states that with respect to strong fairness, a predicate can be decided by a CRN if and only if it is semilinear.

As the "what can be computed by CRNs?" question reaches a conclusion, the focus naturally shifts to its "how fast?" counterpart. The latter question is important as the analysis of CRN runtime complexity enables the comparison between different CRN protocols and ultimately guides the search for better ones. Even for CRNs designed to operate on all (strongly) fair executions, the existing runtime analyses assume that reactions are *scheduled stochastically*, namely, according to the Markov process of the standard stochastic model, consistent with having the aforementioned idealized conditions. However, such conditions may well not hold in real wet-lab experiments, where additional factors can significantly affect the order at which reactions proceed [25]. For example, temperature can fluctuate, or molecules may be temporarily unavailable, perhaps sticking to the side of a test tube or reversibly binding to another reactant. Consequently, our work is driven by the following research question: *Is there a meaningful way to quantify the runtime of CRNs when idealized conditions do not necessarily hold?*

The Quest for an Adversarial Runtime Measure. We search for a runtime measure suitable for *adversarially scheduled* executions, namely, executions that are not subject to the constraints of the aforementioned idealized conditions. This is tricky since the adversarial scheduler may generate (arbitrarily) long execution intervals during which no progress can be made, even if those are not likely to be scheduled stochastically. Therefore, the "adversarial runtime measure" should neutralize the devious behavior of the scheduler by ensuring that

[1] To simplify the discussions, we subsequently stick to the CRN terminology even when citing literature that was originally written in terms of these related models.

[2] We follow the common assumption that each reaction involves at most two reactants.

the protocol is not unduly penalized from such bad execution intervals. To guide our search, we look for inspiration from another domain of decentralized computation that faced a similar challenge: distributed network algorithms.

While it is straightforward to measure the runtime of (idealized) synchronous distributed protocols, early on, researchers identified the need to define runtime measures also for (adversarially scheduled) asynchronous distributed protocols [6, 16]. The adversarial runtime measures that were formulated in this regard share the following two principles: (P1) partition the execution into *rounds*, so that in each round, the protocol has an opportunity to make progress; and (P2) calibrate the runtime charged to the individual rounds so that if the adversarial scheduler opts to generate the execution according to the idealized benchmark, then the adversarial runtime measure coincides with the idealized one.

Specifically, in the context of asynchronous message passing protocols, Awerbuch [6] translates principle (P1) to the requirement that no message is delayed for more than a single round, whereas in the context of the distributed daemon, Dolev et al. [16] translate this principle to the requirement that each node is activated at least once in every round. For principle (P2), both [6] and [16] take the "idealized benchmark" to be a synchronous execution in which every round costs one time unit.

When it comes to formulating an adversarial runtime measure for CRN protocols, principle (P2) is rather straightforward: we should make sure that on stochastically generated executions (playing the "idealized benchmark" role), the adversarial runtime measure agrees (in expectation) with that of the corresponding continuous time Markov process. Interpreting principle (P1), however, seems more difficult as it is not clear how to partition the execution into rounds so that in each round, the protocol "has an opportunity to make progress".

The first step towards resolving this difficulty is to introduce an alternative notion of fairness, referred to hereafter as *weak fairness*: An execution is weakly fair if a continuously applicable reaction (i.e., one for which the needed reactants are available) is not starved; such a reaction is either eventually scheduled or the system reaches a configuration where the reaction is inapplicable. Using a graph theoretic characterization, we show that any CRN protocol whose correctness is guaranteed on weakly fair executions is correct also on strongly fair executions (see Cor. 3), thus justifying the weak vs. strong terminology choice. It turns out that for predicate decidability, strong fairness is actually not strictly stronger: protocols operating under the weak fairness assumption can decide all (and only) semilinear predicates (see Thm. 8).

It remains to come up with a scheme that partitions an execution of CRN protocols into rounds in which the weakly fair adversarial scheduler can steer the execution in a nefarious direction, but also the protocol has an opportunity to make progress. A naive attempt at ensuring progress would be to end the current round once every applicable reaction is either scheduled or becomes inapplicable; the resulting partition is too coarse though since in general, a CRN does not have to "wait" for *all* its applicable reactions in order to make progress. Another naive attempt is to end the current round once *any* reaction is scheduled; this yields a partition which is too fine, allowing the scheduler to charge the protocol's run-time for (arbitrarily many) "progress-less rounds".

So, which reaction is necessary for the CRN protocol to make progress? We do not have a good answer for this question, but we know who does...

Runtime and Skipping Policies. Our adversarial runtime measure is formulated so that it is the protocol designer who decides which reaction is necessary for the CRN protocol to make progress. This is done by means of a *runtime policy* ϱ, used solely for the runtime analysis, that maps each configuration $\mathbf{c}$ to a *target* reaction $\varrho(\mathbf{c})$. (Our actual definition of

runtime policies is more general, mapping each configuration to a set of target reactions; see Sec. 4.) Symmetrically to the protocol designer's runtime policy, we also introduce a *skipping policy* σ, chosen by the adversarial scheduler, that maps each step $t \geq 0$ to a step $\sigma(t) \geq t$.

These two policies partition a given execution η into successive rounds based on the following inductive scheme: Round 0 starts at step $t(0) = 0$. Assuming that round $i \geq 0$ starts at step $t(i)$, the prefix of round i is determined by the adversarial skipping policy σ so that it lasts until step $\sigma(t(i))$; let $\mathbf{e}^i$ denote the configuration in step $\sigma(t(i))$, referred to as the round's *effective configuration*. Following that, the suffix of round i is determined by the protocol designer's runtime policy ϱ so that it lasts until the earliest step in which the target reaction $\varrho(\mathbf{e}^i)$ of the round's effective configuration $\mathbf{e}^i$ is either scheduled or becomes inapplicable. That is, in each round, the adversarial scheduler determines (by means of the skipping policy) the round's effective configuration, striving to ensure that progress from this configuration is slow, whereas the runtime policy determines when progress has been made from the effective configuration. This scheme is well defined by the choice of weak fairness; we emphasize that this would not be the case with strong fairness.

The partition of execution η into rounds allows us to ascribe a runtime to η by charging each round with a *temporal cost* and then accumulating the temporal costs of all rounds until η terminates.[3] The temporal cost of round i is defined to be the expected (continuous) time until the target reaction $\varrho(\mathbf{e}^i)$ of its effective configuration $\mathbf{e}^i$ is either scheduled or becomes inapplicable in an imaginary execution that starts at $\mathbf{e}^i$ and proceeds according to the stochastic scheduler.[4] In other words, the protocol's runtime is *not* charged for the prefix of round i that lasts until the (adversarially chosen) effective configuration is reached; the temporal cost charged for the round's suffix, emerging from the effective configuration, is the expected time that this suffix would have lasted in a stochastically scheduled execution (i.e., the idealized benchmark).

The asymptotic runtime of the CRN protocol is defined by minimizing over all runtime policies ϱ and then maximizing over all weakly fair executions η and skipping policies σ. Put differently, the protocol designer first commits to ϱ and only then, the (weakly fair) adversarial scheduler determines η and σ.

Intuitively, the challenge in constructing a good runtime policy ϱ (the challenge one faces when attempting to up-bound a protocol's runtime) is composed of two, often competing, objectives (see, e.g., Fig. 1): On the one hand, $\varrho(\mathbf{c})$ should be selected so that every execution η is guaranteed to gain "significant progress" by the time a round whose effective configuration is $\mathbf{c}$ ends, thus minimizing the number of rounds until η terminates. On the other hand, $\varrho(\mathbf{c})$ should be selected so that the temporal cost of such a round is small, thus minimizing the contribution of the individual rounds to η's runtime. In the typical scenarios, a good runtime policy ϱ results in partitioning η into $n^{\Theta(1)}$ rounds, each contributing a temporal cost between $\Theta(1/n)$ and $\Theta(n)$, where n is η's initial molecular count.

To verify that our adversarial runtime measure is indeed compatible with the aforementioned principle (P2), we show that if the (adversarial) scheduler opts to generate the execution η stochastically, then our runtime measure coincides (in expectation) with that of the corresponding continuous time Markov process (see Lem. 5). The adversarial scheduler however can be more malicious than that: simple examples show that in general, the runtime of a CRN protocol on adversarially scheduled executions may be significantly larger than on stochastically scheduled executions (see Fig. 2 and 3).

[3] The exact meaning of termination in this regard is made clear in Sec. 2.

[4] Here, it is assumed that the stochastic scheduler operates with no rate coefficients and with a linear volume (a.k.a. "parallel time"), see Sec. 2.

While runtime analyses of CRNs in the presence of common defect modes can be insightful, a strength of our adversarial model is that it is not tied to specific defects in actual CRNs or their biomolecular implementations. In particular, if the adversarial runtime of a CRN matches its stochastic runtime, then we would expect the CRN to perform according to its stochastic runtime even in the presence of defect modes that we may not anticipate. Moreover, in cases where stochastic runtime analysis is complex (involving reasoning about many different executions of a protocol and their likelihoods), it may in fact be easier to determine the adversarial runtime since it only requires stochastic analysis from rounds' effective configurations. For similar reasons, notions of adversarial runtime have proven to be valuable in design of algorithms in both centralized and decentralized domains more broadly, even when they do not capture realistic physical scenarios. Finally, while the analysis task of finding a good runtime policy for a given CRN may seem formidable at first, our experience in analyzing the protocols in this paper is that such a runtime policy is quite easy to deduce, mirroring intuition about the protocol's strengths and weaknesses.

The Runtime of Predicate Decidability. After formulating the new adversarial runtime measure, we turn our attention to CRN protocols whose goal is to decide whether the initial configuration satisfies a given predicate, indicated by the presence of designated Boolean ("yes" and "no") *voter* species in the output configuration. As mentioned earlier, the predicates that can be decided in that way are exactly the semilinear predicates, which raises the following two questions: What is the optimal adversarial runtime of protocols that decide semilinear predicates in general? Are there semilinear predicates that can be decided faster?

A notion that plays an important role in answering these questions is that of CRN *speed faults*, introduced in the impressive work of Chen et al. [10]. This notion captures a (reachable) configuration from which any path to an output configuration includes a (bimolecular) reaction both of whose reactants appear in $O(1)$ molecular counts. The significance of speed faults stems from the fact that any execution that reaches such a "pitfall configuration" requires $\Omega(n)$ time (in expectation) to terminate under the standard stochastic model.[5] The main result of [10] states that a predicate can be decided by a speed fault free CRN protocol (operating under the strongly fair adversarial scheduler) if and only if it belongs to the class of detection predicates (a subclass of semilinear predicates).

The runtime measure introduced in the current paper can be viewed as a quantitative generalization of the fundamentally qualitative notion of speed faults (the quest for such a generalization was, in fact, the main motivation for this work). As discussed in Sec. 4.1, in our adversarial setting, a speed fault translates to an $\Omega(n)$ runtime lower bound, leading to an $\Omega(n)$ runtime lower bound for the task of deciding any non-detection semilinear predicate. On the positive side, we prove that this bound is tight: any semilinear predicate (in particular, the non-detection ones) can be decided by a CRN protocol operating under the weakly fair adversarial scheduler whose runtime is $O(n)$ (see Thm. 8). For detection predicates, we establish a better upper bound (which is also tight): any detection predicate can be decided by a CRN protocol operating under the weakly fair adversarial scheduler whose runtime is $O(\log n)$ (see Thm. 9). Refer to Table 1 for a summary of the adversarial runtime complexity bounds established for predicate decidability tasks; for comparison, Table 2 presents a similar summary of the known stochastic runtime complexity bounds.

[5] The definition of runtime in [10] is based on a slightly different convention which results in scaling the runtime expressions by a $1/n$ factor.

Paper's Outline. The rest of the paper is organized as follows. The CRN model used in this paper is presented in Sec. 2. In Sec. 3, we link the correctness of a CRN protocol to certain topological properties of its configuration digraph. Our new runtime notion for adversarially scheduled executions is introduced in Sec. 4, where we also establish the soundness of this notion and formalize its connection to speed faults. Sec. 5 presents our results for predicate deciding CRNs. These are accompanied by a generic technique for amplifying the molecular count of the voter species in the outcome, introduced in Sec. 6.

2 Chemical Reaction Networks

In this section, we present the *chemical reaction network (CRN)* computational model. For the most part, we adhere to the conventions of the existing CRN literature (e.g., [14, 12, 9]), but we occasionally deviate from them for the sake of simplifying the subsequent discussions. (Refer to Fig. 1a–4a for illustrations of the notions presented in this section.)

A *CRN* is a protocol Π specified by the pair $\Pi = (\mathcal{S}, \mathcal{R})$, where $\mathcal{S}$ is a fixed set of *species* and $\mathcal{R} \subset \mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}$ is a fixed set of *reactions*.[6] For a reaction $\alpha = (\mathbf{r}, \mathbf{p}) \in \mathcal{R}$, the vectors $\mathbf{r} \in \mathbb{N}^{\mathcal{S}}$ and $\mathbf{p} \in \mathbb{N}^{\mathcal{S}}$ specify the stoichiometry of α's *reactants* and *products*, respectively.[7] Specifically, the entry $\mathbf{r}(A)$ (resp., $\mathbf{p}(A)$) indexed by a species $A \in \mathcal{S}$ in the vector $\mathbf{r}$ (resp., $\mathbf{p}$) encodes the number of molecules of A that are consumed (resp., produced) when α is applied. Species A is a *catalyst* for the reaction $\alpha = (\mathbf{r}, \mathbf{p})$ if $\mathbf{r}(A) = \mathbf{p}(A) > 0$.

We adhere to the convention (see, e.g., [11, 17, 15, 10]) that each reaction $(\mathbf{r}, \mathbf{p}) \in \mathcal{R}$ is either *unimolecular* with $\|\mathbf{r}\| = 1$ or *bimolecular* with $\|\mathbf{r}\| = 2$. [8] Note that if all reactions $(\mathbf{r}, \mathbf{p}) \in \mathcal{R}$ are bimolecular and *density preserving*, namely, $\|\mathbf{r}\| = \|\mathbf{p}\|$, then the CRN model is equivalent to the extensively studied *population protocols* model [2, 5, 22] assuming that the population protocol agents have a constant state space.

For a vector (or multiset) $\mathbf{r} \in \mathbb{N}^{\mathcal{S}}$ with $1 \leq \|\mathbf{r}\| \leq 2$, let $\mathcal{R}(\mathbf{r}) = (\{\mathbf{r}\} \times \mathbb{N}^{\mathcal{S}}) \cap \mathcal{R}$ denote the subset of reactions whose reactants correspond to $\mathbf{r}$. In the current paper, it is required that none of these reaction subsets is empty, i.e., $|\mathcal{R}(\mathbf{r})| \geq 1$ for every $\mathbf{r} \in \mathbb{N}^{\mathcal{S}}$ with $1 \leq \|\mathbf{r}\| \leq 2$. Some of the reactions in $\mathcal{R}$ may be *void*, namely, reactions $(\mathbf{r}, \mathbf{p})$ satisfying $\mathbf{r} = \mathbf{p}$; let $\mathrm{NV}(\mathcal{R}) = \{(\mathbf{r}, \mathbf{p}) \in \mathcal{R} \mid \mathbf{r} \neq \mathbf{p}\}$ denote the set of non-void reactions in $\mathcal{R}$. To simplify the exposition, we assume that if $\alpha = (\mathbf{r}, \mathbf{r}) \in \mathcal{R}$ is a void reaction, then $\mathcal{R}(\mathbf{r}) = \{\alpha\}$; this allows us to describe protocol Π by listing only its non-void reactions. For the sake of simplicity, we further assume that $\|\mathbf{r}\| \leq \|\mathbf{p}\|$ for all reactions $(\mathbf{r}, \mathbf{p}) \in \mathcal{R}$.

Configurations. A *configuration* of a CRN $\Pi = (\mathcal{S}, \mathcal{R})$ is a vector $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ that encodes the *molecular count* $\mathbf{c}(A)$ of species A in the solution for each $A \in \mathcal{S}$.[9] The molecular count notation is extended to species (sub)sets $\Lambda \subseteq \mathcal{S}$, denoting $\mathbf{c}(\Lambda) = \sum_{A \in \Lambda} \mathbf{c}(A)$. We refer to $\mathbf{c}(\mathcal{S}) = \|\mathbf{c}\|$ as the *molecular count* of the configuration $\mathbf{c}$. Let $\mathbf{c}|_{\Lambda} \in \mathbb{N}^{\Lambda}$ denote the restriction of a configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ to a species subset $\Lambda \subseteq \mathcal{S}$.

A reaction $\alpha = (\mathbf{r}, \mathbf{p}) \in \mathcal{R}$ is said to be *applicable* to a configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ if $\mathbf{r}(A) \leq \mathbf{c}(A)$ for every $A \in \mathcal{S}$. Let $\mathrm{app}(\mathbf{c}) \subseteq \mathcal{R}$ denote the set of reactions which are applicable to $\mathbf{c}$ and let $\overline{\mathrm{app}}(\mathbf{c}) = \mathcal{R} - \mathrm{app}(\mathbf{c})$, referring to the reactions in $\overline{\mathrm{app}}(\mathbf{c})$ as being *inapplicable* to $\mathbf{c}$. We

[6] Throughout this paper, we denote $\mathbb{N} = \{z \in \mathbb{Z} \mid z \geq 0\}$.
[7] We stick to the convention of identifying vectors in $\mathbb{N}^{\mathcal{S}}$ with multisets over $\mathcal{S}$ expressed as a "molecule summation".
[8] The notation $\|\cdot\|$ denotes the 1-norm ℓ_1.
[9] We consider the discrete version of the CRN model, where the configuration encodes integral molecular counts, in contrast to the continuous model, where a configuration is given by real species densities.

restrict our attention to configurations $\mathbf{c}$ with molecular count $\|\mathbf{c}\| \geq 1$, which ensures that $\text{app}(\mathbf{c})$ is never empty. For a reaction $\alpha \in \text{app}(\mathbf{c})$, let $\alpha(\mathbf{c}) = \mathbf{c} - \mathbf{r} + \mathbf{p}$ be the configuration obtained by applying α to $\mathbf{c}$.[10]

Given two configurations $\mathbf{c}, \mathbf{c}' \in \mathbb{N}^{\mathcal{S}}$, the binary relation $\mathbf{c} \rightharpoonup \mathbf{c}'$ holds if there exists a reaction $\alpha \in \text{app}(\mathbf{c})$ such that $\alpha(\mathbf{c}) = \mathbf{c}'$. We denote the reflexive transitive closure of $\rightharpoonup$ by $\overset{*}{\rightharpoonup}$ and say that $\mathbf{c}'$ is *reachable* from $\mathbf{c}$ if $\mathbf{c} \overset{*}{\rightharpoonup} \mathbf{c}'$. Given a configuration set $Z \subseteq \mathbb{N}^{\mathcal{S}}$, let

$$\text{stab}(Z) \triangleq \left\{\mathbf{c} \in Z \mid \mathbf{c} \overset{*}{\rightharpoonup} \mathbf{c}' \implies \mathbf{c}' \in Z\right\} \quad \text{and} \quad \text{halt}(Z) \triangleq \left\{\mathbf{c} \in Z \mid \mathbf{c} \overset{*}{\rightharpoonup} \mathbf{c}' \implies \mathbf{c}' = \mathbf{c}\right\},$$

observing that the latter set is a (not necessarily strict) subset of the former.

For the sake of simplicity, we restrict this paper's focus to protocols that *respect finite density* [17], namely, $\mathbf{c} \overset{*}{\rightharpoonup} \mathbf{c}'$ implies that $\|\mathbf{c}'\| \leq O(\|\mathbf{c}\|)$. We note that density preserving CRNs inherently respect finite density, however we also allow for reactions that have more products than reactants as long as the CRN protocol is designed so that the molecular count cannot increase arbitrarily. This means, in particular, that although the configuration space $\mathbb{N}^{\mathcal{S}}$ is inherently infinite, the set $\{\mathbf{c}' \in \mathbb{N}^{\mathcal{S}} \mid \mathbf{c} \overset{*}{\rightharpoonup} \mathbf{c}'\}$ is finite (and bounded as a function of $\|\mathbf{c}\|$) for any configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$.

Executions. An *execution* η of the CRN Π is an infinite sequence $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t \geq 0}$ of $\langle \text{configuration}, \text{reaction} \rangle$ pairs such that $\alpha^t \in \text{app}(\mathbf{c}^t)$ and $\mathbf{c}^{t+1} = \alpha^t(\mathbf{c}^t)$ for every $t \geq 0$. It is convenient to think of η as progressing in discrete *steps* so that configuration $\mathbf{c}^t$ and reaction α^t are associated with step $t \geq 0$. We refer to $\mathbf{c}^0$ as the *initial configuration* of η and, unless stated otherwise, denote the molecular count of $\mathbf{c}^0$ by $n = \|\mathbf{c}^0\|$. Given a configuration set $Z \subseteq \mathbb{N}^{\mathcal{S}}$, we say that η *stabilizes* (resp., *halts*) into Z if there exists a step $t \geq 0$ such that $\mathbf{c}^t \in \text{stab}(Z)$ (resp., $\mathbf{c}^t \in \text{halt}(Z)$) and refer to the earliest such step t as the execution's *stabilization step* (resp., *halting step*) with respect to Z.

In this paper, we consider an *adversarial scheduler* that knows the CRN protocol Π and the initial configuration $\mathbf{c}^0$ and determines the execution $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t \geq 0}$ in an arbitrary (malicious) way. The execution η is nonetheless subject to the following *fairness* condition: for every $t \geq 0$ and for every $\alpha \in \text{app}(\mathbf{c}^t)$, there exists $t' \geq t$ such that either (I) $\alpha^{t'} = \alpha$; or (II) $\alpha \notin \text{app}(\mathbf{c}^{t'})$. In other words, the scheduler is not allowed to (indefinitely) "starve" a continuously applicable reaction. We emphasize that the mere condition that a reaction $\alpha \in \mathcal{R}$ is applicable infinitely often does not imply that α is scheduled infinitely often.

Note that the fairness condition adopted in the current paper differs from the one used in the existing CRN literature [2, 4, 11, 9]. The latter, referred to hereafter as *strong fairness*, requires that if a configuration $\mathbf{c}$ appears infinitely often in the execution η and a configuration $\mathbf{c}'$ is reachable from $\mathbf{c}$, then $\mathbf{c}'$ also appears infinitely often in η. Strictly speaking, a strongly fair execution η is not necessarily fair according to the current paper's notion of fairness (in particular, η may starve void reactions). However, as we show in Sec. 3, protocol correctness under the current paper's notion of fairness implies protocol correctness under strong fairness (see Cor. 3), where the exact meaning of correctness is defined soon. Consequently, we refer hereafter to the notion of fairness adopted in the current paper as *weak fairness*.

Interface and Correctness. The CRN notions introduced so far are independent of any particular computational task. To correlate between a CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ and concrete computational tasks, we associate Π with a (task specific) *interface* $\mathcal{I} = (\mathcal{U}, \mu, \mathcal{C})$ whose

[10] Unless stated otherwise, all vector arithmetic is done component-wise.

semantics is as follows: $\mathcal{U}$ is a fixed set of *interface values* that typically encode the input and/or output associated with the species; $\mu : \mathcal{S} \to \mathcal{U}$ is an *interface mapping* that maps each species $A \in \mathcal{S}$ to an interface value $\mu(A) \in \mathcal{U}$; and $\mathcal{C} \subseteq \mathbb{N}^{\mathcal{U}} \times \mathbb{N}^{\mathcal{U}}$ is a *correctness relation* that determines the correctness of an execution as explained soon.[11]

Hereafter, we refer to the vectors in $\mathbb{N}^{\mathcal{U}}$ as *interface vectors*. The interface of a configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ in terms of the input/output that $\mathbf{c}$ encodes is captured by the interface vector

$$\mu(\mathbf{c}) \triangleq \langle \mathbf{c}\left(\{A \in \mathcal{S} \mid \mu(A) = u\}\right)\rangle_{u \in \mathcal{U}} .$$

The abstract interface $\mathcal{I} = (\mathcal{U}, \mu, \mathcal{C})$ allows us to define what it means for a protocol to be correct. To this end, for each configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$, let $Z_{\mathcal{I}}(\mathbf{c}) = \{\mathbf{c}' \in \mathbb{N}^{\mathcal{S}} \mid (\mu(\mathbf{c}), \mu(\mathbf{c}')) \in \mathcal{C}\}$ be the set of configurations which are mapped by μ to interface vectors that satisfy the correctness relation with $\mu(\mathbf{c})$. A configuration $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$ is a *valid initial configuration* with respect to $\mathcal{I}$ if $Z_{\mathcal{I}}(\mathbf{c}^0) \neq \emptyset$; an execution is *valid* (with respect to $\mathcal{I}$) if it emerges from a valid initial configuration. A valid execution η is said to be *stably correct* (resp., *haltingly correct*) if η stabilizes (resp., halts) into $Z_{\mathcal{I}}(\mathbf{c}^0)$. The protocol Π is said to be *stably correct* (resp., *haltingly correct*) if every weakly fair valid execution is guaranteed to be stably (resp., haltingly) correct.[12]

The Stochastic Scheduler. While the current paper focuses on the (weakly fair) adversarial scheduler, another type of scheduler that receives a lot of attention in the literature is the *stochastic scheduler*. Here, we present the stochastic scheduler so that it can serve as a "benchmark" for the runtime definition introduced in Sec. 4. To this end, we define the *propensity* of a reaction $\alpha = (\mathbf{r}, \mathbf{p}) \in \mathcal{R}$ in a configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$, denoted by $\pi_{\mathbf{c}}(\alpha)$, as

$$\pi_{\mathbf{c}}(\alpha) = \begin{cases} \mathbf{c}(A) \cdot \frac{1}{|\mathcal{R}(\mathbf{r})|} , & \mathbf{r} = A \\ \frac{1}{\varphi} \cdot \binom{\mathbf{c}(A)}{2} \cdot \frac{1}{|\mathcal{R}(\mathbf{r})|} , & \mathbf{r} = 2A \\ \frac{1}{\varphi} \cdot \mathbf{c}(A) \cdot \mathbf{c}(B) \cdot \frac{1}{|\mathcal{R}(\mathbf{r})|} , & \mathbf{r} = A + B, A \neq B \end{cases} ,$$

where $\varphi > 0$ is a (global) *volume* parameter.[13] Notice that reaction α is applicable to $\mathbf{c}$ if and only if $\pi_{\mathbf{c}}(\alpha) > 0$. The propensity notation is extended to reaction (sub)sets $Q \subseteq \mathcal{R}$ by defining $\pi_{\mathbf{c}}(Q) = \sum_{\alpha \in Q} \pi_{\mathbf{c}}(\alpha)$. Recalling that $\mathcal{R}(\mathbf{r}) \neq \emptyset$ for each $\mathbf{r} \in \mathbb{N}^{\mathcal{S}}$ with $1 \leq \|\mathbf{r}\| \leq 2$, we observe that

$$\pi_{\mathbf{c}} \triangleq \pi_{\mathbf{c}}(\mathcal{R}) = \|\mathbf{c}\| + \frac{1}{\varphi} \cdot \binom{\|\mathbf{c}\|}{2} .$$

The stochastic scheduler determines the execution $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t \geq 0}$ by scheduling a reaction $\alpha \in \mathrm{app}(\mathbf{c}^t)$ in step t, setting $\alpha^t = \alpha$, with probability proportional to α's propensity $\pi_{\mathbf{c}^t}(\alpha)$ in $\mathbf{c}^t$. The assumption that the CRN protocol respects finite density implies that η is (weakly and strongly) fair with probability 1. We define the *time span* of step $t \geq 0$ to be $1/\pi_{\mathbf{c}^t}$,

[11] The abstract interface formulation generalizes various families of computational tasks addressed in the CRN literature, including predicate decision [8, 10, 9] (see also Sec. 5) and function computation [11, 18, 9], as well as the vote amplification task discussed in Sec. 6, without committing to the specifics of one particular family. For example, for the CRDs presented in Sec. 5, we define $\mathcal{U} = (\Sigma \cup \{\bot\}) \times \{0, 1, \bot\}$. The interface mapping μ then maps each species $A \in \mathcal{S}$ to the interface value $\mu(A) = (x, y) \in \mathcal{U}$ defined so that (I) $x = A$ if $A \in \Sigma$; and $x = \bot$ otherwise; and (II) $y = v$ if $A \in \Upsilon_v$; and $y = \bot$ otherwise.

[12] Both notions of correctness have been studied in the CRN literature, see, e.g., [9].

[13] In the standard stochastic model [20], the propensity expression is multiplied by a reaction specific rate coefficient. In the current paper, that merely uses the stochastic scheduler as a benchmark, we make the simplifying assumption that all rate coefficients are set to 1 (c.f. [11, 10]).

i.e., the normalizing factor of the reaction selection probability.[14] Given a step $t^* \geq 0$, the *stochastic runtime* of the execution prefix $\eta^* = \langle \mathbf{c}^t, \alpha^t \rangle_{0 \leq t < t^*}$ is defined to be the accumulated time span $\sum_{t=0}^{t^*-1} 1/\pi_{\mathbf{c}^t}$.

We adopt the convention that the volume is proportional to the initial molecular count $n = \|\mathbf{c}^0\|$ [17]. The assumption that the CRN protocol respects finite density ensures that $\varphi = \Theta(\|\mathbf{c}^t\|)$ for every $t \geq 0$ which means that the volume is sufficiently large to contain all molecules throughout the (stochastic) execution η. This also means that the time span of each step $t \geq 0$ is

$$1/\pi_{\mathbf{c}^t} = \frac{\varphi}{\varphi \cdot \|\mathbf{c}^t\| + \binom{\|\mathbf{c}^t\|}{2}} = \Theta(1/\|\mathbf{c}^t\|) = \Theta(1/n)\,, \tag{1}$$

hence the stochastic runtime of an execution prefix that lasts for t^* steps is $\Theta(t^*/n)$.

3 Correctness Characterization via the Configuration Digraph

It is often convenient to look at CRN protocols through the lens of the following abstract directed graph (a.k.a. digraph): The *configuration digraph* of a protocol $\Pi = (\mathcal{S}, \mathcal{R})$ is a digraph, denoted by D^Π, whose edges are labeled by reactions in $\mathcal{R}$. The nodes of D^Π are identified with the configurations in $\mathbb{N}^\mathcal{S}$; the edge set of D^Π includes an α-labeled edge from $\mathbf{c}$ to $\alpha(\mathbf{c})$ for each configuration $\mathbf{c} \in \mathbb{N}^\mathcal{S}$ and reaction $\alpha \in \mathrm{app}(\mathbf{c})$ (thus the outdegree of $\mathbf{c}$ in D^Π is $|\mathrm{app}(\mathbf{c})|$). Observe that the self-loops of D^Π are exactly the edges labeled by (applicable) void reactions. Moreover, a configuration $\mathbf{c}'$ is reachable, in the graph theoretic sense, from a configuration $\mathbf{c}$ if and only if $\mathbf{c} \xrightarrow{*} \mathbf{c}'$. For a configuration $\mathbf{c} \in \mathbb{N}^\mathcal{S}$, let $D^\Pi_\mathbf{c}$ be the digraph induced by D^Π on the set of configurations reachable from $\mathbf{c}$ and observe that $D^\Pi_\mathbf{c}$ is finite as Π respects finite density. (Refer to Fig. 1b–4b for illustrations of the notions presented in this section.)

The *(strongly connected) components* of the configuration digraph D^Π are the equivalence classes of the "reachable from each other" relation over the configurations in $\mathbb{N}^\mathcal{S}$. We say that a reaction $\alpha \in \mathcal{R}$ *escapes* from a component S of D^Π if every configuration in S admits an outgoing α-labeled edge to a configuration not in S; i.e., $\alpha \in \mathrm{app}(\mathbf{c})$ and $\alpha(\mathbf{c}) \notin S$ for every $\mathbf{c} \in S$ (see, e.g., Fig. 1b). The notion of escaping reactions allows us to express the stable/halting correctness of CRNs in terms of their configuration digraphs.

▶ **Lemma 1.** *A CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ is stably (resp., haltingly) correct with respect to an interface $\mathcal{I} = (\mathcal{U}, \mu, \mathcal{C})$ under a weakly fair scheduler if and only if for every valid initial configuration $\mathbf{c}^0 \in \mathbb{N}^\mathcal{S}$, every component S of $D^\Pi_{\mathbf{c}^0}$ satisfies (at least) one of the following two conditions: (1) S admits some (at least one) escaping reaction; or (2) $S \subseteq \mathrm{stab}(Z_\mathcal{I}(\mathbf{c}^0))$ (resp., $S \subseteq \mathrm{halt}(Z_\mathcal{I}(\mathbf{c}^0))$), where $Z_\mathcal{I}(\mathbf{c}^0) = \{\mathbf{c} \in \mathbb{N}^\mathcal{S} \mid (\mu(\mathbf{c}^0), \mu(\mathbf{c})) \in \mathcal{C}\}$.*

To complement Lem. 1, we also express the stable/halting correctness of CRNs in terms of their configuration digraphs under a strongly fair scheduler.

▶ **Lemma 2.** *A CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ is stably (resp., haltingly) correct with respect to an interface $\mathcal{I} = (\mathcal{U}, \mu, \mathcal{C})$ under a strongly fair scheduler if and only if for every valid initial configuration $\mathbf{c}^0 \in \mathbb{N}^\mathcal{S}$, every component S of $D^\Pi_{\mathbf{c}^0}$ satisfies (at least) one of the following two conditions: (1) S admits some (at least one) edge outgoing to another component; or (2) $S \subseteq \mathrm{stab}(Z_\mathcal{I}(\mathbf{c}^0))$ (resp., $S \subseteq \mathrm{halt}(Z_\mathcal{I}(\mathbf{c}^0))$), where $Z_\mathcal{I}(\mathbf{c}^0) = \{\mathbf{c} \in \mathbb{N}^\mathcal{S} \mid (\mu(\mathbf{c}^0), \mu(\mathbf{c})) \in \mathcal{C}\}$.*

[14] The time span definition is consistent with the expected time until a reaction occurs under the continuous time Markov process formulation of the standard stochastic model [20] with no rate coefficients.

Combining Lem. 1 and 2, we obtain the following corollary.

▶ **Corollary 3.** *If a CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ is stably (resp., haltingly) correct with respect to an interface $\mathcal{I}$ under a weakly fair scheduler, then Π is also stably (resp., haltingly) correct with respect to $\mathcal{I}$ under a strongly fair scheduler.*

4 The Runtime of Adversarially Scheduled Executions

So far, the literature on CRN protocols operating under an adversarial scheduler focused mainly on computability, leaving aside, for the most part, complexity considerations. [15] This is arguably unavoidable when working with the strong fairness condition which is inherently oblivious to the chain of reactions that realizes the reachability of one configuration from another. In the current paper, however, we adopt the weak fairness condition which facilitates the definition of a quantitative measure for the runtime of adversarially scheduled executions, to which this section is dedicated. (Refer to Fig. 1c–4c for illustrations of the notions presented in this section.)

Consider a stably (resp., haltingly) correct CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$. We make extensive use of the following operator: Given a weakly fair execution $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t \geq 0}$, a step $t \geq 0$, and a reaction (sub)set $Q \subseteq \mathcal{R}$, let $\tau(\eta, t, Q)$ be the earliest step $s > t$ such that at least one of the following two conditions is satisfied:
(I) $\alpha^{s-1} \in Q$; or
(II) $Q \subseteq \bigcup_{t \leq t' \leq s} \overline{\mathrm{app}}(\mathbf{c}^{t'})$.
(This operator is well defined by the weak fairness of η.)

Intuitively, we think of the operator $\tau(\eta, t, Q)$ as a process that tracks η from step t onward and stops once any Q reaction is scheduled (condition (I)). This by itself is not well defined as the scheduler may avoid scheduling the Q reactions from step t onward. However, the scheduler must prevent the starvation of any continuously applicable reaction in Q, so we also stop the τ-process once the adversary "fulfills this commitment" (condition (II)).

The Policies. Our runtime measure is based on partitioning a given weakly fair execution $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t \geq 0}$ into *rounds*. This is done by means of two policies: a *runtime policy* ϱ, determined by the protocol designer, that maps each configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ to a non-void reaction (sub)set $\varrho(\mathbf{c}) \subseteq \mathrm{NV}(\mathcal{R})$, referred to as the *target reaction set* of $\mathbf{c}$ under ϱ; and a *skipping policy* σ, determined by the adversarial scheduler (in conjunction with the execution η), that maps each step $t \geq 0$ to a step $\sigma(t) \geq t$.

Round $i = 0, 1, \ldots$ spans the step interval $[t(i), t(i+1))$ and includes a designated *effective step* $t(i) \leq t_{\mathrm{e}}(i) < t(i+1)$. The partition of execution η into rounds is defined inductively by setting

$$t(i) \;=\; \begin{cases} 0\,, & i = 0 \\ \tau\left(\eta, t_{\mathrm{e}}(i-1), \varrho\left(\mathbf{c}^{t_{\mathrm{e}}(i-1)}\right)\right)\,, & i > 0 \end{cases} \qquad \text{and} \qquad t_{\mathrm{e}}(i) \;=\; \sigma(t(i))\,.$$

Put differently, for every round $i \geq 0$ with initial step $t(i)$, the adversarial scheduler first determines the round's effective step $t_{\mathrm{e}}(i) = \sigma(t(i)) \geq t(i)$ by means of the skipping policy σ. Following that, we apply the runtime policy ϱ (chosen by the protocol designer) to the configuration $\mathbf{e}^i = \mathbf{c}^{t_{\mathrm{e}}(i)}$, referred to as the round's *effective configuration*, and obtain the

[15] The one exception in this regard is the work of Chen et al. [10] on speed faults – see Sec. 4.1 and 5.

target reaction set $Q = \varrho(\mathbf{e}^i)$. The latter is then plugged into the operator τ to determine $t(i+1) = \tau(\eta, t_e(i), Q)$. Round i is said to be *target-accomplished* if $\alpha^{t(i+1)-1} \in Q$; otherwise, it is said to be *target-deprived*.

▶ Remark. Our definition of the runtime policy ϱ does not require that the reactions included in the target reaction set $\varrho(\mathbf{c})$ are applicable to the configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$. Notice though that if all target reactions are inapplicable to $\mathbf{c}$ (which is bound to be the case if $\mathbf{c}$ is halting), then a round whose effective configuration is $\mathbf{c}$ is destined to be target deprived and end immediately after the effective step, regardless of the reaction scheduled in that step. In the full version [13], we investigate several other "natural restrictions" of the runtime policy definition, including fixed policies and singleton target reaction sets, showing that they all lead to significant efficiency loss.

Temporal Cost. We define the *temporal cost* of a configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ under a runtime policy ϱ, denoted by $\mathrm{TC}^{\varrho}(\mathbf{c})$, as follows: Let $\eta_r = \langle \mathbf{c}_r^t, \alpha_r^t \rangle_{t \geq 0}$ be a stochastic execution emerging from the initial configuration $\mathbf{c}_r^0 = \mathbf{c}$ and define

$$\mathrm{TC}^{\varrho}(\mathbf{c}) \triangleq \mathbb{E}\left(\sum_{t=0}^{\tau(\eta_r,0,\varrho(\mathbf{c}))-1} 1/\pi_{\mathbf{c}_r^t}\right) = \Theta(1/\|\mathbf{c}\|) \cdot \mathbb{E}\left(\tau(\eta_r, 0, \varrho(\mathbf{c}))\right) ,$$

where the expectation is over the random choice of η_r and the second transition is due to (1). That is, the temporal cost of $\mathbf{c}$ under ϱ is defined to be the expected stochastic runtime of round 0 of η_r with respect to the runtime policy ϱ and the identity skipping policy σ_{id} that maps each step $t \geq 0$ to $\sigma_{\mathrm{id}}(t) = t$ (which means that the effective step of each round is its initial step).

Execution Runtime. Consider a runtime policy ϱ and a skipping policy σ. Let $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t \geq 0}$ be a weakly fair valid execution and let $t(i)$, $t_e(i)$, and $\mathbf{e}^i = \mathbf{c}^{t_e(i)}$ be the initial step, effective step, and effective configuration, respectively, of round $i \geq 0$ under ϱ and σ. Fix some step $t^* \geq 0$ and consider the execution prefix $\eta^* = \langle \mathbf{c}^t, \alpha^t \rangle_{0 \leq t < t^*}$. We define the *(adversarial) runtime* of η^* under ϱ and σ, denoted by $\mathrm{RT}^{\varrho,\sigma}(\eta^*)$, by taking $i^* = \min\{i \geq 0 \mid t(i) \geq t^*\}$ and setting

$$\mathrm{RT}^{\varrho,\sigma}(\eta^*) \triangleq \sum_{i=0}^{i^*-1} \mathrm{TC}^{\varrho}\left(\mathbf{e}^i\right) .$$

The *stabilization runtime* (resp., *halting runtime*) of the (entire) execution η under ϱ and σ, denoted by $\mathrm{RT}^{\varrho,\sigma}_{\mathrm{stab}}(\eta)$ (resp., $\mathrm{RT}^{\varrho,\sigma}_{\mathrm{halt}}(\eta)$), is defined to be $\mathrm{RT}^{\varrho,\sigma}\left(\langle \mathbf{c}^t, \alpha^t \rangle_{0 \leq t < t^*}\right)$, where $t^* \geq 0$ is the stabilization (resp., halting) step of η. In other words, we use ϱ and σ to partition η into rounds and mark the effective steps. Following that, we charge each round i that starts before step t^* according to the temporal cost (under ϱ) of its effective configuration $\mathbf{e}^i$.

Looking at it from another angle, using the skipping policy σ, the adversarial scheduler determines the sequence $\mathbf{e}^0, \mathbf{e}^1, \ldots$ of effective configurations according to which the temporal cost $\mathrm{TC}^{\varrho}(\mathbf{e}^i)$ of each round $i \geq 0$ is calculated. By choosing an appropriate runtime policy ϱ, the protocol designer may (1) ensure that progress is made from one effective configuration to the next, thus advancing η towards round $i^* = \min\{i \geq 0 \mid t(i) \geq t^*\}$; and (2) bound the contribution $\mathrm{TC}^{\varrho}(\mathbf{e}^i)$ of each round $0 \leq i < i^*$ to the stabilization runtime $\mathrm{RT}^{\varrho,\sigma}_{\mathrm{stab}}(\eta)$ (resp., halting runtime $\mathrm{RT}^{\varrho,\sigma}_{\mathrm{halt}}(\eta)$). The crux of our runtime definition is that this contribution depends only on the effective configuration $\mathbf{e}^i$, irrespectively of how round i actually develops (see, e.g., Fig. 1c).

▶ Remark. Using this viewpoint, it is interesting to revisit the definitions of [6] and [16] for the runtime of an asynchronous distributed protocol $\mathcal{P}$. Following the discussion in Sec. 1, this runtime is defined as the length of the longest sequence $\mathbf{e}^0, \mathbf{e}^1, \ldots, \mathbf{e}^{i^*-1}$ of "non-terminal"

configurations (of $\mathcal{P}$) such that $\mathbf{e}^i$ is reachable from $\mathbf{e}^{i-1}$ by an execution interval that lasts for at least one round (according to the respective definitions of [6] and [16]). Our adversarial runtime is defined in the same manner, taking $\mathbf{e}^0, \mathbf{e}^1, \ldots, \mathbf{e}^{i^*-1}$ to be the first i^* effective (CRN) configurations, only that we charge each configuration $\mathbf{e}^i$ according to its temporal cost (rather than one "runtime unit" as in [6] and [16]). This difference is consistent with the different "idealized benchmarks": a synchronous schedule in [6] and [16] vs. a stochastic execution in the current paper. The skipping policy σ plays a key role in adversarially generating the sequence $\mathbf{e}^0, \mathbf{e}^1, \ldots, \mathbf{e}^{i^*-1}$ as it "decouples" between the last step of round i, determined by the runtime policy ϱ, and the effective configuration $\mathbf{e}^{i+1}$ of round $i+1$ (see, e.g., Fig. 4c).

The Runtime Function. For $n \geq 1$, let $\mathcal{F}(n)$ denote the set of weakly fair valid executions $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t \geq 0}$ of initial molecular count $\|\mathbf{c}^0\| = n$. The *stabilization runtime* (resp., *halting runtime*) of the CRN protocol Π for executions in $\mathcal{F}(n)$, denoted by $\mathrm{RT}^{\Pi}_{\mathrm{stab}}(n)$ (resp., $\mathrm{RT}^{\Pi}_{\mathrm{halt}}(n)$), is defined to be

$$\mathrm{RT}^{\Pi}_{\mathrm{x}}(n) \triangleq \min_{\varrho} \max_{\eta \in \mathcal{F}(n),\, \sigma} \mathrm{RT}^{\varrho,\sigma}_{\mathrm{x}}(\eta)\,,$$

where x serves as a placeholder for stab (resp., halt). This formalizes the responsibility of the protocol designer to specify a runtime policy ϱ, in conjunction with the protocol Π, used for up-bounding Π's stabilization (resp., halting) runtime (see, e.g., Fig. 1c).

The following two lemmas establish the soundness of our adversarial runtime definition: Lem. 4 ensures that the stabilization (resp., halting) runtime function is well defined.[16] In Lem. 5, we show that if the scheduler generates the execution stochastically, then our (adversarial) runtime measure agrees in expectation with the stochastic runtime measure.

▶ **Lemma 4.** *Consider a stably (resp., haltingly) correct protocol $\Pi = (\mathcal{S}, \mathcal{R})$. There exists a runtime policy ϱ such that for every integer $n \geq 1$, execution $\eta \in \mathcal{F}(n)$, and skipping policy σ, the stabilization runtime $\mathrm{RT}^{\varrho,\sigma}_{\mathrm{stab}}(\eta)$ (resp., halting runtime $\mathrm{RT}^{\varrho,\sigma}_{\mathrm{halt}}(\eta)$) is up-bounded as a function of n.*

▶ **Lemma 5.** *Consider a stably (resp., haltingly) correct protocol $\Pi = (\mathcal{S}, \mathcal{R})$. Let $\eta_r = \langle \mathbf{c}^t_r, \alpha^t_r \rangle_{t \geq 0}$ be a stochastic execution emerging from a valid initial configuration $\mathbf{c}^0_r$ and let $t^* \geq 0$ be the stabilization (resp., halting) step of η_r. Then,*

$$\min_{\varrho} \mathbb{E}_{\eta_r} \left(\max_{\sigma} \mathrm{RT}^{\varrho,\sigma}_{\mathrm{x}}(\eta_r) \right) = \mathbb{E}_{\eta_r} \left(\sum_{t=0}^{t^*-1} 1 / \pi_{\mathbf{c}^t_r} \right),$$

where x *serves as a placeholder for* stab *(resp.,* halt*).*

4.1 Speed Faults

Consider a CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ which is stably (resp., haltingly) correct with respect to an interface $\mathcal{I} = (\mathcal{U}, \mu, \mathcal{C})$. For a valid initial configuration $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$, let $Z_{\mathcal{I}}(\mathbf{c}^0) = \{\mathbf{c} \in \mathbb{N}^{\mathcal{S}} \mid (\mu(\mathbf{c}^0), \mu(\mathbf{c})) \in \mathcal{C}\}$ and recall that if a weakly fair execution η of Π emerges from $\mathbf{c}^0$, then η is guaranteed to reach $\mathrm{stab}(Z_{\mathcal{I}}(\mathbf{c}^0))$ (resp., $\mathrm{halt}(Z_{\mathcal{I}}(\mathbf{c}^0))$).

Given a parameter $s > 0$, a configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ is said to be a *stabilization s-pitfall* (resp., a *halting s-pitfall*) of the valid initial configuration $\mathbf{c}^0$ if $\mathbf{c}^0 \overset{*}{\rightharpoonup} \mathbf{c}$ and every path from $\mathbf{c}$ to $\mathrm{stab}(Z_{\mathcal{I}}(\mathbf{c}^0))$ (resp., $\mathrm{halt}(Z_{\mathcal{I}}(\mathbf{c}^0))$) in the digraph D^{Π} includes (an edge labeled by) a

[16] Note that in Lem. 4 we use a universal runtime policy that applies to all choices of the initial molecular count n. This is stronger in principle than what the runtime definition actually requires.

reaction whose propensity is at most s/φ (see, e.g., Fig. 2c and 4c). When $s = O(1)$, we often omit the parameter and refer to $\mathbf{c}$ simply as a *stabilization pitfall* (resp., *halting pitfall*). Following the definition of Chen et al. [10], we say that an infinite family $\mathbf{C}^0$ of valid initial configurations has a *stabilization speed fault* (resp., *halting speed fault*) if for every integer $n_0 > 0$, there exists a configuration $\mathbf{c}^0 \in \mathbf{C}^0$ of molecular count $\|\mathbf{c}^0\| = n \geq n_0$ that admits a stabilization (resp., halting) pitfall.

▶ **Lemma 6.** *If an infinite family $\mathbf{C}^0$ of valid initial configurations has a stabilization (resp., halting) speed fault, then for every integer $n_0 > 0$, there exist a configuration $\mathbf{c}^0 \in \mathcal{C}^0$ of molecular count $\|\mathbf{c}^0\| = n \geq n_0$, a weakly fair execution η emerging from $\mathbf{c}^0$, and a skipping policy σ, such that $\mathrm{RT}_{\mathrm{x}}^{\varrho,\sigma}(\eta) \geq \Omega(n)$ for every runtime policy ϱ, where* x *serves as a placeholder for* stab *(resp.,* halt*).*[17]

5 Predicate Decidability

An important class of CRN protocols is that of *chemical reaction deciders (CRDs)* whose goal is to determine whether the initial molecular counts of certain species satisfy a given predicate. In its most general form (see [10, 9]), a CRD is a CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ augmented with (1) a set $\Sigma \subset \mathcal{S}$ of *input species*; (2) two disjoint sets $\Upsilon_0, \Upsilon_1 \subset \mathcal{S}$ of *voter species*; (3) a designated *fuel species* $F \in \mathcal{S} - \Sigma$; and (4) a fixed *initial context* $\mathbf{k} \in \mathbb{N}^{\mathcal{S}-(\Sigma\cup\{F\})}$. The CRD is said to be *leaderless* if its initial context is the zero vector, i.e., $\mathbf{k} = \mathbf{0}$.

A configuration $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$ is valid as an initial configuration of the CRD Π if $\mathbf{c}^0|_{\mathcal{S}-(\Sigma\cup\{F\})} = \mathbf{k}$; to ensure that the initial molecular count $\|\mathbf{c}^0\|$ is always at least 1 (especially when the CRD is leaderless), we also require that $\mathbf{c}^0(F) \geq 1$. In other words, a valid initial configuration $\mathbf{c}^0$ can be decomposed into an *input vector* $\mathbf{c}^0|_\Sigma = \mathbf{x} \in \mathbb{N}^\Sigma$, the initial context $\mathbf{c}^0|_{\mathcal{S}-(\Sigma\cup\{F\})} = \mathbf{k}$, and any number $\mathbf{c}^0(F) \geq 1$ of fuel molecules. We emphasize that in contrast to the initial context, the protocol designer has no control over the *exact* molecular count of the fuel species in the initial configuration.

For $v \in \{0,1\}$, let $\mathcal{D}_v = \left\{\mathbf{c} \in \mathbb{N}^{\mathcal{S}} \mid \mathbf{c}(\Upsilon_v) > 0 \wedge \mathbf{c}(\Upsilon_{1-v}) = 0\right\}$ be the set of configurations that include v-voters and no $(1-v)$-voters. An input vector $\mathbf{x} \in \mathbb{N}^\Sigma$ is said to be *stably accepted* (resp., *haltingly accepted*) by Π if for every valid initial configuration $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$ with $\mathbf{c}^0|_\Sigma = \mathbf{x}$, every weakly fair execution $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t\geq 0}$ stabilizes (resp., halts) into $\mathcal{D}_1$; the input vector $\mathbf{x} \in \mathbb{N}^\Sigma$ is said to be *stably rejected* (resp., *haltingly rejected*) by Π if the same holds with $\mathcal{D}_0$. The CRD Π is stably (resp., haltingly) correct if every input vector $\mathbf{x} \in \mathbb{N}^\Sigma$ is either stably (resp., haltingly) accepted or stably (resp., haltingly) rejected by Π. In this case, we say that Π *stably decides* (resp., *haltingly decides*) the predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$ defined so that $\psi(\mathbf{x}) = 1$ if and only if $\mathbf{x}$ is stably (resp., haltingly) accepted by Π.

By definition, the molecular count of the fuel species F in the initial configuration $\mathbf{c}^0$ does not affect the computation's outcome in terms of whether the execution stabilizes (resp., halts) with 0- or 1-voters. Consequently, one can increase the molecular count $\mathbf{c}^0(F)$ of the fuel species in the initial configuration $\mathbf{c}^0$, thus increasing the initial (total) molecular count $n = \|\mathbf{c}^0\|$ for any given input vector $\mathbf{x} \in \mathbb{N}^\Sigma$. Since the runtime of a CRN is expressed in terms of the initial molecular count n, decoupling $\mathbf{x}$ from n allows us to measure the asymptotic runtime of the protocol while keeping $\mathbf{x}$ fixed. In this regard, the CRD Π is said to be *stabilization speed fault free* (resp., *halting speed fault free*) [10] if for every input vector $\mathbf{x} \in \mathbb{N}^\Sigma$, the family of valid initial configurations $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$ with $\mathbf{c}^0|_\Sigma = \mathbf{x}$ does not admit a stabilization (resp., halting) speed fault (as defined in Sec. 4.1).

[17] As discussed in [10], a speed fault does not imply an $\Omega(n)$ lower bound on the (stochastic) runtime of stochastically scheduled executions since the probability of reaching a pitfall configuration may be small.

5.1 Semilinear Predicates

A predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$ is *linear* if there exist a finite set $\mathcal{A} = \mathcal{A}(\psi) \subset \mathbb{N}^\Sigma$ and a vector $\mathbf{b} = \mathbf{b}(\psi) \in \mathbb{N}^\Sigma$ such that $\psi(\mathbf{x}) = 1$ if and only if $\mathbf{x} = \mathbf{b} + \sum_{\mathbf{a}\in\mathcal{A}} k_\mathbf{a}\mathbf{a}$ for some coefficients $k_\mathbf{a} = k_\mathbf{a}(\mathbf{x}) \in \mathbb{N}$, $\mathbf{a} \in \mathcal{A}$. A predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$ is *semilinear* if it is the disjunction of finitely many linear predicates. The following theorem is established in the seminal work of Angluin et al. [2, 4].

▶ **Theorem 7** ([2, 4]). *Fix a predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$. If ψ is semilinear, then ψ can be haltingly decided under a strongly fair scheduler by a leaderless CRD. If ψ can be stably decided by a CRD under a strongly fair scheduler, then ψ is semilinear.*

In the full version [13], we extend Thm. 7 to weak fairness which allows us to bound the adversarial runtime of the corresponding CRDs and establish the following theorem; notice that the $O(n)$ runtime bound is asymptotically tight – see the speed fault freeness discussion in Sec. 5.2.

▶ **Theorem 8.** *Fix a predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$. If ψ is semilinear, then ψ can be haltingly decided under a weakly fair scheduler by a leaderless CRD whose halting runtime is $O(n)$. If ψ can be stably decided by a CRD under a weakly fair scheduler, then ψ is semilinear.*

5.2 Detection Predicates

For a vector $\mathbf{x} \in \mathbb{N}^\Sigma$, let $\mathbf{x}_\downarrow \in \{0,1\}^\Sigma \subset \mathbb{N}^\Sigma$ be the vector defined so that $\mathbf{x}_\downarrow(A) = 1 \iff \mathbf{x}(A) > 0$. A predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$ is a *detection* predicate if $\psi(\mathbf{x}) = \psi(\mathbf{x}_\downarrow)$ for all $\mathbf{x} \in \mathbb{N}^\Sigma$ (cf. [1, 10, 19]). Chen et al. [10] prove that a predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$ can be stably decided under the strongly fair adversarial scheduler by a stabilization speed fault free CRD if and only if it is a detection predicate. Cor. 3 ensures that the only if direction translates to our weakly fair adversarial scheduler; employing Lem. 6, we conclude that a non-detection predicate cannot be decided by a CRD whose stabilization runtime is better than $\Omega(n)$. For the if direction, the construction in [10] yields leaderless CRDs that haltingly decide ψ whose expected halting runtime under the stochastic scheduler is $O(\log n)$. The following theorem states that the same (asymptotic) runtime upper bound can be obtained under the weakly fair adversarial scheduler; the theorem is proved in the full version [13], where we also explain why the promised upper bound is asymptotically tight.

▶ **Theorem 9.** *For every detection predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$, there exists a leaderless CRD that haltingly decides ψ whose halting runtime is $O(\log n)$. Moreover, the CRD is designed so that all molecules in the halting configuration are voters.*

6 Vote Amplification

Recall that CRDs are required to stabilize/halt into configurations $\mathbf{c}$ that include a positive number of v-voter molecules and zero $(1-v)$-voter molecules, where $v \in \{0,1\}$ is determined by the decided predicate according to the input vector. This requirement alone does not rule out the possibility of having a small (yet positive) voter molecular count in $\mathbf{c}$. Indeed, the semilinear predicate CRDs promised in Thm. 8 are designed so that the configuration $\mathbf{c}$ includes a single voter molecule (this is in contrast to the detection predicate CRDs promised in Thm. 9, where all molecules in $\mathbf{c}$ are voters).

In practice though, it may be difficult to obtain a meaningful signal from small molecular counts. Consequently, we aim for *vote amplified* CRDs, namely, CRDs that guarantee to stabilize/halt into configurations in which the voter molecules take all but an ϵ-fraction of

the total molecular count for an arbitrarily small constant $\epsilon > 0$. These are obtained by means of a "generic compiler" that can be applied, in a black-box manner, to any existing CRD, turning it into a vote amplified CRD while preserving the original stabilization/halting correctness. At the heart of this compiler lies a CRN protocol for a standalone computational task, referred to as *vote amplification (VA)*, whose runtime dominates the runtime overhead of the compiler, as stated in the following theorem (proved in the full version [13]).

▶ **Theorem 10.** *Consider a predicate $\psi : \mathbb{N}^\Sigma \to \{0,1\}$ that can be haltingly decided by a (leaderless) CRD in $T_\psi(n)$ time. The existence of a VA protocol that stabilizes (resp., halts) in $T_{\text{amp}}(n)$ time implies the existence of a (leaderless) vote amplified CRD that stably (resp., haltingly) decides ψ in $T_\psi(O(n)) + T_{\text{amp}}(O(n)) + O(\log n)$ time.*

Assuming a stochastic scheduler, Angluin et al. [3] develop a VA protocol that halts in $O(n)$ time. Unfortunately, the protocol of [3] does not meet the topological conditions of Lem. 1, hence the (weakly fair) adversarial scheduler can prevent this protocol from stabilizing (see the full version [13] for more details). Using a completely different technique, we develop a VA protocol whose guarantees are cast in the following theorem.

▶ **Theorem 11.** *There exists a VA protocol (operating under the weakly fair scheduler) that stabilizes in $O(n)$ time and halts in $O(n \log n)$ time.*

Combined with Thm. 10, we obtain a compiler whose stabilization and halting runtime overheads are $O(n)$ and $O(n \log n)$, respectively. Applying this compiler to the CRDs promised in Thm. 8 results in vote amplified CRDs whose stabilization runtime remains $O(n)$, however their halting runtime increases to $O(n \log n)$. The excessive $\log n$ factor would be shaved by a VA protocol that halts in $O(n)$ time whose existence remains an open question.

Task Formalization. A VA protocol is a CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ whose species set $\mathcal{S}$ is partitioned into the pairwise disjoint sets $\mathcal{P}_0 \cup \mathcal{P}_1 \cup \mathcal{F}_0 \cup \mathcal{F}_1 = \mathcal{S}$, where for $v \in \{0,1\}$, the species in $\mathcal{P}_v$ are referred to as *permanent v-voters* and the species in $\mathcal{F}_v$ are referred to as *fluid v-voters*. The permanent voters are regarded as part of the task specification and can participate in the reactions of Π only as catalysts (which means that the molecular count of each permanent voter remains invariant throughout the execution).

A configuration $\mathbf{c}^0 \in \mathbb{N}^\mathcal{S}$ is a valid initial configuration for the VA task if there exists a vote $v \in \{0,1\}$ such that $\mathbf{c}^0(\mathcal{P}_v) > 0$ and $\mathbf{c}^0(\mathcal{P}_{1-v}) = 0$, in which case we refer to $\mathbf{c}^0$ as a *v-voting* initial configuration. A configuration $\mathbf{c} \in \mathbb{N}^\mathcal{S}$ is an *amplification* of a v-voting initial configuration $\mathbf{c}^0$ if (1) $\mathbf{c}(A) = \mathbf{c}^0(A)$ for every $A \in \mathcal{P}_0 \cup \mathcal{P}_1$; (2) $\mathbf{c}(\mathcal{F}_v) = \mathbf{c}^0(\{\mathcal{F}_0 \cup \mathcal{F}_1\})$; and (3) $\mathbf{c}(\mathcal{F}_{1-v}) = 0$. In other words, an amplification of a v-voting initial configuration keeps the original permanent voter molecules and shifts all fluid voter molecules to the v-voting side.

The VA protocol Π is stably (resp., haltingly) correct if every weakly fair valid execution $\eta = \langle \mathbf{c}^t, \alpha^t \rangle_{t \geq 0}$ stabilizes (resp., halts) into the (set of) amplifications of $\mathbf{c}^0$. The typical scenario involves a small number of permanent v-voter molecules and the challenge is to ensure that all fluid voter molecules "end up" in $\mathcal{F}_v$. We emphasize that for Π to be correct, the protocol should handle any initial configuration $\mathbf{c}^0|_{\mathcal{F}_0 \cup \mathcal{F}_1}$ of the fluid voters.

The VA Protocol. We now turn to develop the VA protocol $\Pi = (\mathcal{S}, \mathcal{R})$ promised in Thm. 11. For simplicity, assume in this extended abstract that $\mathcal{P}_0$ and $\mathcal{P}_1$ are singleton sets with $\mathcal{P}_0 = \{P_0\}$ and $\mathcal{P}_1 = \{P_1\}$; the general case is handled in the full version [13]. Protocol Π is defined over the fluid voter sets $\mathcal{F}_0 = \{H_0, L_0\}$ and $\mathcal{F}_1 = \{H_1, L_1\}$. Semantically, we think of the H (resp., L) fluid voters as having a high (resp., low) confidence level in their

vote. The reaction set $\mathcal{R}$ of Π includes the following non-void reactions:
β_v^A: $P_v + A \to P_v + H_v$ for every $v \in \{0,1\}$ and $A \in \{H_{1-v}, L_0, L_1\}$;
γ: $H_0 + H_1 \to L_0 + L_1$; and
δ_v: $H_v + L_{1-v} \to 2L_v$ for every $v \in \{0,1\}$.
Informally, these reactions guarantee that the adversary has little leverage because, as we show soon, *all* of the non-void reactions make nontrivial progress in their own different ways.

For the runtime analysis of protocol Π, consider a weakly fair valid execution $\eta = \langle \mathbf{c}^t, \zeta^t \rangle_{t \geq 0}$ of initial molecular count $\|\mathbf{c}^0\| = n$. Assume for simplicity that the initial configuration $\mathbf{c}^0$ is 1-voting which means that $\mathbf{c}^t(P_1) > 0$ and $\mathbf{c}^t(P_0) = 0$ for all $t \geq 0$; the case where $\mathbf{c}^0$ is 0-voting is analyzed symmetrically. Let $m = \mathbf{c}^0(\{H_0, L_0, L_1, H_1\})$ be the initial molecular count of the fluid voters and observe that $\mathbf{c}^t(\{H_0, L_0, L_1, H_1\}) = m$ for every $t \geq 0$.

To capture progress , we assign an integral score $s(\cdot)$ to each fluid voter by setting $s(H_0) = -4$, $s(L_0) = -1$, $s(L_1) = 1$, and $s(H_1) = 2$. Substituting the $s(\cdot)$ scores into each reaction $\alpha \in \mathrm{NV}(\mathcal{R})$ reveals that the sum of scores of α's fluid reactants is strictly smaller than the sum of scores of α's fluid products. Denoting the total score in a configuration $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ by $s(\mathbf{c}) = \sum_{A \in \{H_0, L_0, L_1, H_1\}} \mathbf{c}(A) \cdot s(A)$, we deduce that $s(\mathbf{c}^{t+1}) \geq s(\mathbf{c}^t)$ and that $\zeta^t \in \mathrm{NV}(\mathcal{R}) \implies s(\mathbf{c}^{t+1}) > s(\mathbf{c}^t)$ for every $t \geq 0$. Since $-4m \leq s(\mathbf{c}^t) \leq 2m$ for every $t \geq 0$, it follows that η includes, in total, at most $O(m) \leq O(n)$ non-void reactions until it stabilizes.

The last bound ensures that progress is made on each non-void reaction. Accordingly, we choose the runtime policy ϱ so that $\varrho(\mathbf{c}) = \mathrm{NV}(\mathcal{R})$ for all configurations $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$.[18]

Fix some skipping policy σ and let $\mathbf{e}^i$ be the effective configuration of round $i \geq 0$ under ϱ and σ. Let $i^* = \min\{i \geq 0 \mid \mathbf{e}^i(\{H_0, L_0\}) = 0\}$ be the first round whose effective step appears after η stabilizes. Since the choice of ϱ ensures that each round $0 \leq i < i^*$ is target-accomplished, ending with a non-void reaction, it follows that $i^* \leq O(n)$.

To bound the stabilization runtime of execution η under ϱ and σ, we argue that $\pi_{\mathbf{e}^i}(\mathrm{NV}(\mathcal{R})) \geq \Omega(1)$ for every $0 \leq i < i^*$; by a simple probabilistic argument (elaborated in the full version [13]), this allows us to conclude that $\mathrm{TC}^{\varrho}(\mathbf{e}^i) \leq O(1)$ for every $0 \leq i < i^*$. To this end, notice that if $\mathbf{e}^i(H_1) \geq m/2$, then

$$\pi_{\mathbf{e}^i}(\{\gamma, \delta_1\}) = \tfrac{1}{\varphi} \cdot \mathbf{e}^i(H_1) \cdot \mathbf{e}^i(\{H_0, L_0\}) \geq \Omega(m/n) = \Omega(1)\,.$$

Otherwise ($\mathbf{e}^i(H_1) < m/2$), we know that $\mathbf{e}^i(\{H_0, L_0, L_1\}) > m/2$, hence

$$\pi_{\mathbf{e}^i}\left(\left\{\beta_1^A \mid A \in \{H_0, L_0, L_1\}\right\}\right) = \tfrac{1}{\varphi} \cdot \mathbf{e}^i(\{H_0, L_0, L_1\}) \cdot \mathbf{e}^i(\mathcal{P}_1) \geq \Omega(m/n) = \Omega(1)\,,$$

thus establishing the argument. Therefore, the stabilization runtime of η satisfies

$$\mathrm{RT}^{\varrho,\sigma}_{\mathrm{stab}}(\eta) = \textstyle\sum_{i=0}^{i^*-1} \mathrm{TC}^{\varrho}(\mathbf{e}^i) \leq \sum_{i=0}^{O(n)} O(1) = O(n)\,.$$

The proof of Thm. 11 is completed by showing that protocol Π halts in $O(n \log n)$ time. This part of the proof is deferred to the full version [13].

References

1 Dan Alistarh, Bartłomiej Dudek, Adrian Kosowski, David Soloveichik, and Przemysław Uznański. Robust detection in leak-prone population protocols. In *DNA Computing and Molecular Programming: 23rd International Conference, DNA 23, Austin, TX, USA, September 24–28, 2017, Proceedings 23*, pages 155–171. Springer, 2017.

[18] Although it serves its purpose in the current analysis, for many CRN protocols, a runtime policy whose targets cover all non-void reactions is suboptimal; this is elaborated in the full version [13].

2 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006.

3 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008.

4 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distrib. Comput.*, 20(4):279–304, 2007.

5 James Aspnes and Eric Ruppert. An introduction to population protocols. In Benoît Garbinato, Hugo Miranda, and Luís E. T. Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer, 2009.

6 Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, October 1985. doi:10.1145/4221.4227.

7 Hamid Bolouri and James M. Bower. *Computational Modeling of Genetic and Biochemical Networks*. The MIT Press, February 2001. doi:10.7551/mitpress/2018.001.0001.

8 Robert Brijder. Minimal output unstable configurations in chemical reaction networks and deciders. *Nat. Comput.*, 15(2):235–244, 2016.

9 Robert Brijder. Computing with chemical reaction networks: a tutorial. *Nat. Comput.*, 18(1):119–137, 2019.

10 Ho-Lin Chen, Rachel Cummings, David Doty, and David Soloveichik. Speed faults in computation by chemical reaction networks. *Distributed Comput.*, 30(5):373–390, 2017.

11 Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Nat. Comput.*, 13(4):517–534, 2014.

12 Anne Condon. On design and analysis of chemical reaction network algorithms. In Cezar Câmpeanu, editor, *Implementation and Application of Automata - 23rd International Conference, CIAA 2018, Charlottetown, PE, Canada, July 30 - August 2, 2018, Proceedings*, volume 10977 of *Lecture Notes in Computer Science*, pages 1–3. Springer, 2018.

13 Anne Condon, Yuval Emek, and Noga Harlev. On the runtime of crns beyond idealized conditions, 2023. URL: http://arxiv.org/abs/2307.00647.

14 Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. In Anne Condon, David Harel, Joost N. Kok, Arto Salomaa, and Erik Winfree, editors, *Algorithmic Bioprocesses*, Natural Computing Series, pages 543–584. Springer, 2009.

15 Rachel Cummings, David Doty, and David Soloveichik. Probability 1 computation with chemical reaction networks. *Nat. Comput.*, 15(2):245–261, 2016.

16 S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, 1997. doi:10.1109/71.588622.

17 David Doty. Timing in chemical reaction networks. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 772–784. SIAM, 2014.

18 David Doty and Monir Hajiaghayi. Leaderless deterministic chemical reaction networks. *Nat. Comput.*, 14(2):213–223, 2015.

19 Bartłomiej Dudek and Adrian Kosowski. Universal protocols for information dissemination using emergent signals. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 87–99, 2018.

20 Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

21 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, May 1969. doi:10.1016/S0022-0000(69)80011-5.

22 Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. *New Models for Population Protocols*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2011.

23 Grzegorz Rozenberg and Joost Engelfriet. *Elementary net systems*, pages 12–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. doi:10.1007/3-540-65306-6_14.

24 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Nat. Comput.*, 7(4):615–633, 2008.

25 Marko Vasić, Cameron Chalk, Austin Luchsinger, Sarfraz Khurshid, and David Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.

A FIGURES AND TABLES

Table 1 The (adversarial) runtime complexity landscape of predicate decidability CRN protocols operating under the weakly fair adversarial scheduler. The upper bounds (O-notation) hold with a universal quantifier over the predicate family and an existential quantifier over the CRD family; the lower bounds (Ω-notation) hold with a universal quantifier over both the predicate and CRD families. (As usual, $\Theta(f(n))$ should be interpreted as both $O(f(n))$ and $\Omega(f(n))$.)

predicates	leaderless	amplified vote	stabilization runtime	halting runtime
semilinear (non-detection)	yes	yes	$\Theta(n)$	$\Omega(n)$, $O(n \log n)$
	yes	no	$\Theta(n)$	$\Theta(n)$
	no	yes	$\Theta(n)$	$\Omega(n)$, $O(n \log n)$
	no	no	$\Theta(n)$	$\Theta(n)$
detection	yes	yes	$\Theta(\log n)$	$\Theta(\log n)$
	yes	no	$\Theta(\log n)$	$\Theta(\log n)$
	no	yes	$\Theta(\log n)$	$\Theta(\log n)$
	no	no	$\Theta(\log n)$	$\Theta(\log n)$

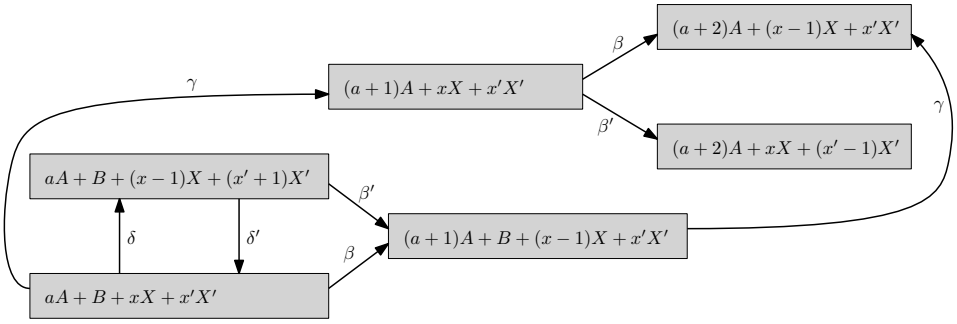
Table 2 The (expected stochastic) runtime complexity landscape of predicate decidability CRN protocols operating under the stochastic scheduler (refer to the full version [13] for details). The upper bounds (O-notation) hold with a universal quantifier over the predicate family and an existential quantifier over the CRD family; the lower bounds (Ω-notation) hold with a universal quantifier over both the predicate and CRD families. (As usual, $\Theta(f(n))$ should be interpreted as both $O(f(n))$ and $\Omega(f(n))$.)

predicates	leaderless	amplified vote	stabilization runtime	halting runtime
semilinear (non-eventually constant)	yes	yes	$\Theta(n)$	$\Theta(n)$
	yes	no	$\Theta(n)$	$\Theta(n)$
	no	yes	$\Omega(\log n)$, $O(n)$	$\Omega(\log n)$, $O(n)$
	no	no	$\Omega(\log n)$, $O(n)$	$\Omega(\log n)$, $O(n)$
eventually constant (non-detection)	yes	yes	$\Omega(\log n)$, $O(n)$	$\Omega(\log n)$, $O(n)$
	yes	no	$\Omega(\log n)$, $O(n)$	$\Omega(\log n)$, $O(n)$
	no	yes	$\Omega(\log n)$, $O(n)$	$\Omega(\log n)$, $O(n)$
	no	no	$\Omega(\log n)$, $O(n)$	$\Omega(\log n)$, $O(n)$
detection	yes	yes	$\Theta(\log n)$	$\Theta(\log n)$
	yes	no	$\Theta(\log n)$	$\Theta(\log n)$
	no	yes	$\Theta(\log n)$	$\Theta(\log n)$
	no	no	$\Theta(\log n)$	$\Theta(\log n)$

$$\mathcal{S} = \{A, B, X, X'\}$$

$$\begin{aligned}
&\beta\colon\ A + X \to 2A\\
&\beta'\colon\ A + X' \to 2A\\
&\gamma\colon\ A + B \to 2A\\
&\delta\colon\ B + X \to B + X'\\
&\delta'\colon\ B + X' \to B + X
\end{aligned}$$

(a) The species and non-void reactions of Π. A configuration $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$ is valid as an initial configuration of Π if $\mathbf{c}^0(A) = \mathbf{c}^0(B) = 1$. The protocol is designed so that the molecular count of A is non-decreasing, whereas the molecular counts of B and of $\{X, X'\}$ are non-increasing. Moreover, all weakly fair valid executions of Π halt in a configuration that includes only A molecules.



(b) Part of the configuration digraph D^{Π} (excluding void reactions). The configuration set $S = \{aA + B + (x+z)X + (x'-z)X' \mid -x \le z \le x'\}$ forms a component of D^{Π}. Since reactions β and β' are inapplicable in configurations $aA + B + (x+x')X'$ and $aA + B + (x+x')X$, respectively, it follows that these two reactions do not escape from S. Reaction γ on the other hand does escape from S, ensuring that a weakly fair execution cannot remain in S indefinitely. After the B molecule is consumed (by a γ reaction), each configuration constitutes a (singleton) component of D^{Π} and every applicable non-void reaction is escaping.

$$\varrho(\mathbf{c}) = \begin{cases} \{\beta, \beta', \gamma\}, & \mathbf{c}(B) > 0 \\ \{\beta, \beta'\}, & \mathbf{c}(B) = 0 \end{cases}$$

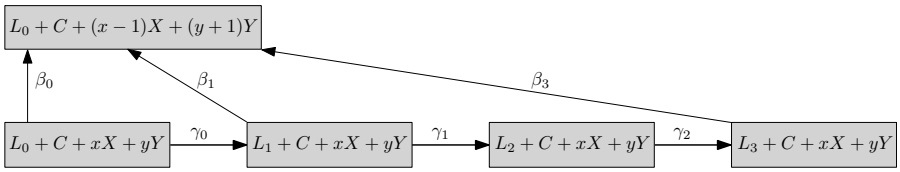
(c) A runtime policy ϱ for Π. Under ϱ, a round with effective configuration $\mathbf{e} = aA + B + xX + x'X'$ is target-accomplished and ends upon scheduling one of the reactions β, β', γ; in particular, it is guaranteed that the next effective configuration $\mathbf{e}'$ satisfies $\mathbf{e}'(A) > a$ (no matter what the adversarial skipping policy is), i.e., progress is made. The adversary may opt to schedule reactions δ and δ' many times before the round ends, however the crux of our runtime definition is that this does not affect the round's temporal cost. Specifically, since $\pi_{\mathbf{c}}(\{\beta, \beta', \gamma\}) = \frac{a(x+x'+1)}{\varphi}$ for every $\mathbf{c} \in S$ (recall the definition of component S from Fig. 1b), it follows, by a simple probabilistic argument, that $\mathrm{TC}^{\varrho}(\mathbf{e}) = \frac{\varphi}{a(x+x'+1)}$. A similar (simpler in fact) argument leads to the conclusion that if $\mathbf{e} = aA + xX + x'X'$, then $\mathrm{TC}^{\varrho}(\mathbf{e}) = \frac{\varphi}{a(x+x')}$. This allows us to show that $\mathrm{RT}^{\Pi}_{\mathrm{halt}}(n) = O(\log n)$.

Figure 1 A CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ demonstrating how a carefully chosen runtime policy guarantees significant progress in each round while up-bounding the round's temporal cost.

$\mathcal{S} = \{L_0, L_1, \ldots, L_{k+1}, C, X, Y\}$

β_i: $L_i + X \to L_0 + Y$ for $0 \le i \le k+1$, $i \neq k$
γ_i: $L_i + C \to L_{i+1} + C$ for $0 \le i \le k$

(a) The species and non-void reactions of Π, where k is an arbitrarily large constant. A configuration $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$ is valid as an initial configuration of Π if $\mathbf{c}^0(L_0) = 1$, $\mathbf{c}^0(C) = 1$, and $\mathbf{c}^0(\{L_1, \ldots, L_{k+1}\}) = 0$. The protocol is designed so that $\mathbf{c}(\{L_0, L_1, \ldots, L_{k+1}\}) = 1$ for any configuration $\mathbf{c}$ reachable from a valid initial configuration (i.e., $L_0, L_1, \ldots, L_{k+1}$ are "leader species"). Species C is a catalyst for any reaction it participates in and $\mathbf{c}(C) = 1$ for any configuration $\mathbf{c}$ reachable from a valid initial configuration. The execution progresses by shifting all X molecules into Y molecules. We are interested in the stabilization of Π's executions into the (set of) configurations $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ satisfying $\mathbf{c}(X) < \mathbf{c}(Y)$, although the executions actually halt once $\mathbf{c}(X) = 0$ (and $\mathbf{c}(L_{k+1}) = 1$).



(b) Part of the configuration digraph D^{Π} (excluding void reactions) for $k = 2$. Notice that each configuration constitutes a (singleton) component of D^{Π} and every applicable non-void reaction is escaping.

$$\varrho(\mathbf{c}) = \mathrm{app}(\mathbf{c}) \cap \mathrm{NV}(\mathcal{R})$$

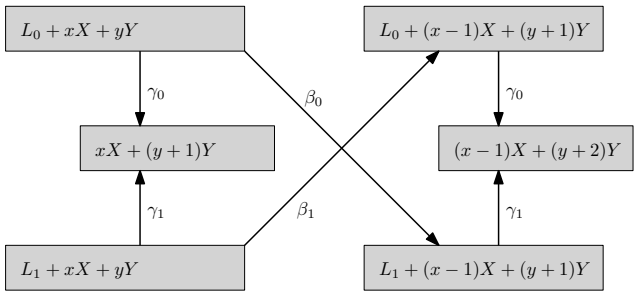
(c) A runtime policy ϱ for Π. Under ϱ, every round ends once any non-void reaction is applied to the round's effective configuration. In the full version [13], we show that $\mathrm{RT}^{\varrho,\sigma}_{\mathrm{stab}}(\eta) \le O(n^2)$ for any skipping policy σ and weakly fair valid execution η of initial molecular count n. It turns out that this bound is tight: The (weakly fair) adversarial scheduler can generate the execution $\eta = \langle \mathbf{c}^t, \alpha^t \rangle$ by scheduling $\alpha^t = \gamma_i$ if $\mathbf{c}^t(L_i) = 1$ for some $0 \le i \le k$; and $\alpha^t = \beta_{k+1}$ if $\mathbf{c}^t(L_{k+1}) = 1$. Using the identity skipping policy σ_{id} and assuming that $\mathbf{c}^0(X) = x_0$ and $\mathbf{c}^0(Y) = 0$, it is easy to show (see the full version [13]) that η visits the configuration $\mathbf{c}_y = L_k + C + (x_0 - y)X + yY$ for every $0 \le y \le x_0/2$ before it stabilizes and that each such visit constitutes the effective configuration of the corresponding round, regardless of the runtime policy ϱ'. Since each such configuration $\mathbf{c}_y$ is a 2-pitfall (recall the definition from Sec. 4.1), we deduce that $\mathrm{TC}^{\varrho'}(\mathbf{c}_y) = \Omega(n)$, which sums up to $\mathrm{RT}^{\varrho',\sigma_{\mathrm{id}}}_{\mathrm{stab}}(\eta) = \Omega(n^2)$. The interesting aspect of protocol Π is that with high probability, a stochastic execution stabilizes without visiting the pitfall configurations $\mathbf{c}_y$ even once, which allows us to conclude that the expected stochastic stabilization runtime of Π is $O(n)$ – see the full version [13] for details.

Figure 2 A CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ demonstrating that the adversarial stabilization runtime may be significantly larger than the expected stochastic runtime due to (asymptotically many) pitfall configurations.

$$\mathcal{S} = \{L_0, L_1, X, Y\}$$

$$\begin{aligned}
\beta_0&\colon\ L_0 + Y \to 2Y\\
\beta_1&\colon\ L_1 + Y \to 2Y\\
\gamma_0&\colon\ L_0 + X \to L_1 + Y\\
\gamma_1&\colon\ L_1 + X \to L_0 + Y
\end{aligned}$$

(a) The species and non-void reactions of Π. A configuration $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$ is valid as an initial configuration of Π if $\mathbf{c}^0(\{L_0, L_1\}) = 1$ and $\mathbf{c}^0(Y) \geq \mathbf{c}^0(X)$. The protocol is designed so that the molecular counts of $\{L_0, L_1\}$ and of X are non-increasing, whereas the molecular count of Y is non-decreasing. Moreover, all weakly fair valid executions of Π halt in a configuration that includes zero L_0 and L_1 molecules.



(b) Part of the configuration digraph D^{Π} (excluding void reactions). Notice that each configuration constitutes a (singleton) component of D^{Π} and every applicable non-void reaction is escaping. Furthermore, if $\mathbf{c} \rightharpoonup \mathbf{c}'$, then $\mathrm{app}(\mathbf{c}) \cap \mathrm{app}(\mathbf{c}') \cap \mathrm{NV}(\mathcal{R}) = \emptyset$.

$$\varrho(\mathbf{c}) = \begin{cases} \beta_0, & \mathbf{c}(L_0) = 1 \\ \beta_1, & \mathbf{c}(L_1) = 1 \end{cases}$$

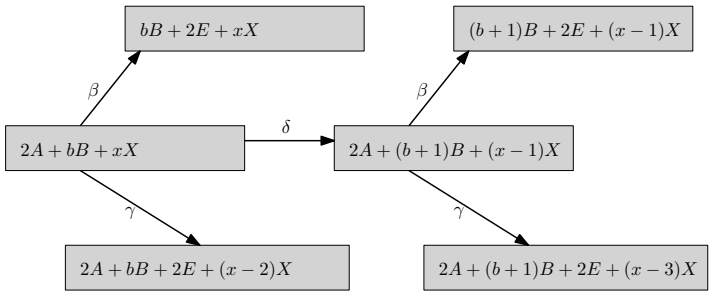
(c) A runtime policy ϱ for Π. Under ϱ, a round whose (non-halting) effective configuration is $\mathbf{e}$ ends once the execution reaches any configuration $\mathbf{c} \neq \mathbf{e}$. This may result from scheduling the target reaction $\varrho(\mathbf{e})$, in which case the round is target-accomplished; otherwise, the round ends because $\varrho(\mathbf{e})$ is inapplicable in $\mathbf{c}$, in which case the round is target-deprived. Since $\pi_{L_i+xX+yY}(\beta_i) = y/\varphi$ for $i \in \{0,1\}$, it follows, by a simple probabilistic argument, that $\mathrm{TC}^{\varrho}(L_i + xX + yY) = \varphi/y$. This allows us to show that $\mathrm{RT}^{\Pi}_{\mathrm{halt}}(n) = O(n)$. In the full version [13], it is shown that the $O(n)$ halting runtime bound is (asymptotically) tight under the (weakly fair) adversarial scheduler; in contrast, the expected stochastic runtime of Π under the stochastic scheduler is only $O(1)$. This gap holds despite the fact that the executions that realize the $\Omega(n)$ runtime lower bound do not reach a pitfall configuration.

Figure 3 A CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ demonstrating that the adversarial runtime may be significantly larger than the expected stochastic runtime even though the protocol does not admit a speed fault.

$$\mathcal{S} = \{A, B, E, X\}$$

$$\begin{aligned}
&\beta\colon\ A + A \to 2E \\
&\gamma\colon\ X + X \to 2E \\
&\delta\colon\ B + X \to 2B \\
&\chi_Z\colon\ E + Z \to 2E \qquad \text{for every } Z \in \{A, B, X\}
\end{aligned}$$

(a) The species and non-void reactions of Π. A configuration $\mathbf{c}^0 \in \mathbb{N}^{\mathcal{S}}$ is valid as an initial configuration of Π if $\mathbf{c}^0(A) = 2$, $\mathbf{c}^0(B) = 1$, and $\mathbf{c}^0(E) = 0$. The protocol is designed so that once the first (two) E molecules are produced, species E spreads by means of the χ reactions until it takes over the entire test tube. The first E molecules can be produced either through reaction β, that remains applicable as long as E was not produced, or through reaction γ, that may become inapplicable if reaction δ is scheduled many times. In any case, all weakly fair valid executions of Π halt in a configuration that includes only E molecules.



(b) Part of the configuration digraph D^{Π} (excluding void reactions). Notice that each configuration constitutes a (singleton) component of D^{Π} and every applicable non-void reaction is escaping.

$$\varrho(\mathbf{c}) = \begin{cases} \{\beta, \gamma\}, & \mathbf{c}(E) = 0 \\ \{\chi_A, \chi_B, \chi_X\}, & \mathbf{c}(E) > 0 \end{cases}$$

(c) A runtime policy ϱ for Π. Under ϱ, a round whose (non-halting) effective configuration $\mathbf{e}$ satisfies $\mathbf{e}(E) = 0$ ends once (two) E molecules are produced, either through reaction β or through reaction γ (either way, the round is target accomplished); in particular, there can be at most one such round, that is, the first round of the execution. To maximize the temporal cost charged for this round, the adversarial scheduler devises (the execution and) the skipping policy σ so that $\mathbf{e}(X) < 2$ which means that reaction γ is inapplicable in $\mathbf{e}$ (this requires that σ generates a "large skip"). Such a configuration $\mathbf{e}$ is a halting 2-pitfall as reaction β must be scheduled in order to advance the execution and the propensity of β is $2/\varphi$. We conclude, by a simple probabilistic argument, that $\mathrm{TC}^{\varrho}(\mathbf{e}) = \varphi/2$. A round whose (non-halting) effective configuration $\mathbf{e}$ satisfies $\mathbf{e}(E) > 0$ ends once a χ configuration is scheduled, thus ensuring that the execution's next effective configuration $\mathbf{e}'$ satisfies $\mathbf{e}'(E) > \mathbf{e}(E)$. Using standard arguments, one can prove that the total contribution of (all) these rounds to the halting runtime of an execution η with initial molecular count n is up-bounded by $O(\log n)$. Therefore, together with the contribution of the first "slow" round, we get $\mathrm{RT}^{\varrho,\sigma}_{\mathrm{halt}}(\eta) = \Theta(n)$. Note that the same runtime policy ϱ leads to a much better halting runtime of $\mathrm{RT}^{\varrho,\sigma_{\mathrm{id}}}_{\mathrm{halt}}(\eta) = \Theta(\log n)$ if the adversarial scheduler opts to use the identity skipping policy σ_{id} instead of the aforementioned skipping policy σ. As a direct consequence of Lem. 6, we conclude that ϱ is asymptotically optimal for Π, hence $\mathrm{RT}^{\Pi}_{\mathrm{halt}}(n) = \Theta(n)$.

Figure 4 A CRN protocol $\Pi = (\mathcal{S}, \mathcal{R})$ demonstrating that a non-trivial skipping policy results in a significantly larger runtime, compared to the identity skipping policy.

Rational Design of DNA Sequences with Non-Orthogonal Binding Interactions

Joseph Don Berleant[1]
Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

Molecular computation involving promiscuous, or non-orthogonal, binding interactions between system components is found commonly in natural biological systems, as well as some proposed human-made molecular computers. Such systems are characterized by the fact that each computational unit, such as a domain within a DNA strand, may bind to several different partners with distinct, prescribed binding strengths. Unfortunately, implementing systems of molecular computation that incorporate non-orthogonal binding is difficult, because researchers lack a robust, general-purpose method for designing molecules with this type of behavior. In this work, we describe and demonstrate a process for the rational design of DNA sequences with prescribed non-orthogonal binding behavior. This process makes use of a model that represents large sets of non-orthogonal DNA sequences using fixed-length binary strings, and estimates the differential binding affinity between pairs of sequences through the Hamming distance between their corresponding binary strings. The real-world applicability of this model is supported by simulations and some experimental data. We then select two previously described systems of molecular computation involving non-orthogonal interactions, and apply our sequence design process to implement them using DNA strand displacement. Our simulated results on these two systems demonstrate both digital and analog computation. We hope that this work motivates the development and implementation of new computational paradigms based on non-orthogonal binding.

2012 ACM Subject Classification Hardware → Emerging architectures; Hardware → Biology-related information processing

Keywords and phrases DNA sequence design, binding networks, promiscuous binding, non-orthogonal binding, isometric graph embeddings

Digital Object Identifier 10.4230/LIPIcs.DNA.29.4

Funding Research supported in part by a National Science Foundation Graduate Research Fellowship (Grant No. 1122374). Research supported in part by the Office of Naval Research (N00014-21-1-4013), the Army Research Office (ICB Subaward W911NF-19-2-0026), and the National Science Foundation (CBET-1729397, OAC-1940231, CCF-1956054).

1 Introduction

The vast majority of prior work with DNA-based molecular computation has dealt with the design and use of orthogonal DNA domains, which are intended to bind only to their perfect complements and to exhibit minimal cross-talk interactions with other domains in a system. This eases the design and analysis of large DNA-based molecular circuits, because it reduces the number of interactions that must be considered, allowing the scalable implementation of circuits with many more components [14, 20, 23]. Comparatively little research has attempted to solve the problem of designing non-orthogonal DNA sequences, that is, sets of DNA domains such that each sequence can bind to several partners with prescribed binding affinities. This gap in our abilities exists despite several well-documented examples of naturally occurring biological networks that make use of non-orthogonal binding

[1] Corresponding author. E-mail address: jberlean@mit.edu

behavior [1,15], computing problems that would benefit from non-orthogonal sequence design like similarity search in large databases [2,17], and nanostructure fabrication strategies that rely on promiscuous binding between structural subunits [16,30].

Very recently, some researchers have attempted to use machine learning for the design of non-orthogonal sequence designs, for example for DNA memory storage [3,27], while others have used exhaustive searches or evolutionary approaches to design non-orthogonal DNA sequences for digital logic and analog computing circuits [18]. While promising, these attempts have been characterized by design inaccuracies and, in some cases, the need for extensive experimental trial and error. Further, some preliminary evidence suggests that computational predictions of strand binding thermodynamics may be less accurate for sequences that are highly non-complementary, although additional data to support this observation is warranted [18]. Currently, the field of molecular computing lacks a general-purpose and efficient method for non-orthogonal DNA sequence design, capable of handling the variety of potential use cases in DNA-based molecular circuits, nanostructure fabrication, and data storage.

In this work, we describe a method for the rational design of non-orthogonal DNA sequences, which is based on two main contributions: the identification of a subset of DNA sequence space for which a simple model accurately predicts binding affinities between pairs of DNA sequences, and the application of a process called isometric graph embedding to the sequence design process. In Sections 3 and 4, we demonstrate our design process in the context of two previously described molecular computation systems involving non-orthogonal interactions [1,18]. Through simulations of these systems, we demonstrate both digital and analog computation using our sequence designs.[2]

2 Rational design of non-orthogonal DNA domains

Our ultimate goal will be to design for the differential binding affinities between pairs of DNA sequences, that is, we will consider the quantities $\Delta G(x, y^*) - \Delta G(w, z^*)$ for DNA sequences x, y, w, z. In Section 2.1, we define a model for the binding affinity between sequences, which will be applicable to a well-defined subset of the overall DNA sequence space. In Section 2.2, we show how this model enables rational design of DNA sequences via isometric graph embeddings.

2.1 Model of binding between non-orthogonal domains

In general, accurate estimation the binding affinity between two arbitrary DNA strands requires involved computation, and our goal is not to address this task for any two DNA strands. Instead, we first focus on defining a subset D of sequence space with properties that are amenable to efficient and accurate DNA sequence design for prescribed binding affinities. The main idea behind defining D will be to identify a set of substitution mutations that may be applied in any combination to some sequence, and whose cumulative effect on binding affinity is approximately additive. This will justify the model that we introduce at the end of this subsection.

Let $\mathcal{D}_n = \{\mathrm{A}, \mathrm{T}, \mathrm{C}, \mathrm{G}\}^n$ be the set of all n-nt DNA sequences using only canonical nucleotides, where all sequences are written from 5′ to 3′. For an n-nt DNA sequence $x \in \mathcal{D}_n$, its perfect complement is denoted x^*. The nucleotides are referenced by subscripts, so that

[2] This work incorporates material from my doctoral thesis [4], which describes the sequence design process (Section 2) and one of the analog computing gates (part of Section 3).

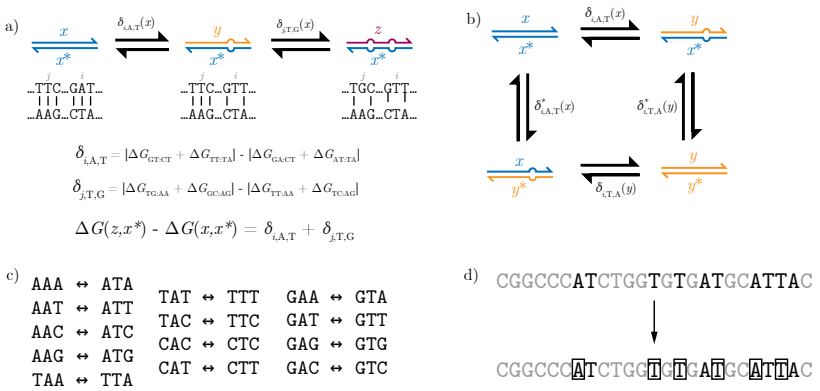


Figure 1 a) The binding affinity of a DNA duplex can be estimated by summing the contributions of each pair adjacent of base pairs. A single point substitution affects the contributions from two of these terms, reflecting the decreased favorability of the two stacking bonds next to a mismatch. A subsequent substitution at a nonadjacent nucleotide will affect two additional terms in the sum. However, under the nearest neighbor model, its effect will be independent of the presence or absence of the first substitution. b) To aid design, we proposed restricting the substitutions to those for which various metrics of its effect on binding affinity agree. For example, mutating the top strand ($\delta_{i,A,T}(x)$) or bottom strand ($\delta^*_{i,A,T}(x)$) should have a similar effect. c) Because the effect of a substitution is determined only by local interactions, we can enumerate all combinations of nucleotides that satisfy our constraints. This makes it very easy to identify sites on a candidate DNA sequence that may be mutated. d) Candidate substitution sites on a DNA site must be filtered to satisfy non-adjacency and interiority requirements.

$x_i \in \{\mathrm{A}, \mathrm{T}, \mathrm{C}, \mathrm{G}\}$ is the i-th nucleotide of x. For $1 \leq i \leq n$ and $a, a' \in \{\mathrm{A}, \mathrm{T}, \mathrm{C}, \mathrm{G}\}$, we define a *substitution* $\sigma_{i,a,a'} : \mathcal{D}_n \to \mathcal{D}_n$ as a function such that $\sigma_{i,a,a'}(x)$ is the sequence generated by substituting the nucleotide a at position i in x with the nucleotide a'.

Fix an $x \in \mathcal{D}_n$, and consider the binding affinity of x to x^*. Under the nearest-neighbor model for DNA strand binding [22], this binding affinity may be estimated from the additive contributions of all pairs of neighboring base pairs in the duplex.[3] For a substitution $\sigma_{i,a,a'}$, $1 < i < n$ and $x_i = a$, let $y = \sigma_{i,a,a'}(x)$. Note that the constraint $1 < i < n$ implies that the substitution occurs at an interior position of x. Under the nearest-neighbor model, the binding affinity of y to x^* differs from that of x to x^* due to the contributions of exactly two affected pairs of adjacent base pairs (Figure 1a).[4] With these assumptions the change in binding affinity $\Delta G(y, x^*) - \Delta G(x, x^*)$ depends only on the values of a, a', x_{i-1}, and x_{i+1}. We will write $\delta_{i,a,a'}(x) := \Delta G(y, x^*) - \Delta G(x, x^*)$, the change in binding affinity from mutating x and leaving x^* unchanged. The corresponding change in binding affinity with x^* mutated and x unchanged will be written $\delta^*_{i,a,a'}(x) := \Delta G(x, y^*) - \Delta G(x, x^*)$. Note that $\delta^*_{i,a,a'}(x) = \delta_{n-i+1,a^*,a'^*}(x^*)$.

For a second substitution $\sigma_{j,b,b'}$, $1 < j < n$ and $x_j = b$, let $z = \sigma_{j,b,b'}(y)$. $\delta_{j,b,b'}(y) = \Delta G(z, y^*) - \Delta G(y, y^*)$ is determined only by the two affected pairs of base pairs. Furthermore, if $|i - j| > 1$, then the base pairs affected by $\sigma_{i,a,a'}$ are disjoint from those affected by $\sigma_{j,b,b'}$, and we may estimate $\Delta G(z, x^*) - \Delta G(x, x^*) = \delta_{i,a,a'} + \delta_{j,b,b'}$. More generally, given a set

[3] This model is biochemically justified by the fact that the principal contributor to binding affinity is the formation of a π-π stacking bond between two adjacent base pairs, and the contribution of each so-called "stack" is determined by the identities of the neighboring nucleotides [22].

[4] This calculation assumes that the substitution occurs in the interior of the sequence, and that "shifted" binding conformations of y to x^* may be neglected. This is likely valid when shifted conformations of x to x^* are unlikely, and the number of substitutions applied is small.

of substitutions occurring at non-adjacent interior positions within x, the effect on binding affinity of applying any subset of these substitutions is the sum of the effects of applying each substitution individually (Figure 1a).

We are now prepared to construct a subset of sequence space $D \in \mathcal{D}_n$. Given $x \in \mathcal{D}_n$, we propose constructing D based on a set of substitutions S satisfying the following requirements:

1. (interiority) For each $\sigma_{i,a,a'} \in S$, $k+1 \leq i \leq n-k$ and $x_i = a \neq a'$.
2. (non-adjacency) For any two $\sigma_{i,a,a'}, \sigma_{j,b,b'} \in S$, $|i-j| > l \geq 1$.
3. (symmetry) For each $\sigma_{i,a,a'} \in S$, $y = \sigma_{i,a,a'}(x)$, the four values $\delta_{i,a,a'}(x)$, $\delta_{i,a',a}(y)$, $\delta^*_{i,a,a'}(x)$, $\delta^*_{i,a',a}(y)$ are within the range $(\delta_{\min}, \delta_{\max})$.

For the interiority requirement, we choose $k = 2$ because of the observation that substitutions at terminal or penultimate strand positions have different effects on binding affinity [19]. For other the non-adjacency requirement, we use $l = 1$ as the minimal requirement for achieving additivity in the effects of each substitution. The symmetry requirement is included to aid subsequent sequence design, and its purpose will become clear by the end of this subsection. Informally, it implies that the effect of a mutation is consistent across different measures of its effect on binding affinity (Figure 1b). Because the effect of mutating a nucleotide depends only on that nucleotide and its neighbors, there are exactly 192 possible substitutions, and we may enumerate the list of those that satisfy condition (3) (Figure 1c). We use the range $(11.0, 15.3)$ kJ/mol, with all binding affinities estimated by published nearest neighbor parameters [22]. In principle, greater design robustness could be achieved by increasing k, increasing l, or decreasing the range $(\delta_{\min}, \delta_{\max})$, at the cost of reducing the size of the set S.

Given a set of substitutions S (Figure 1d), let D be the set of sequences generated by applying a subset $S' \in S$ to x, and D^* be the set of perfect complements to sequences in D. Each element of D is associated with a binary string of length $m := |S|$ as follows. Number the elements of S from 1 to m. Assume without loss of generality that the j-th element of S is a substitution at position i of x. Define $f : D \to \{0,1\}^m$ such that, for $y \in D$, $f(y)$ is the binary string with a j-th bit of 0 if $x_i = y_i$ and 1 otherwise. In other words, a bit of $f(y)$ is 1 if and only if the corresponding substitution in S was applied to generate y from x.

We may now model the differential binding affinity between two pairs of sequences taken from D and D^*. For sequences y, z, u, v from D, we model the differential binding affinity as

$$\Delta G(y, z^*) - \Delta G(u, v^*) \approx \delta_{\text{mut}} \times [d_H(f(y), f(z)) - d_H(f(u), f(v))] \tag{1}$$

where δ_{mut} is a proportionality constant approximating the change in binding affinity due to a single mutation, and d_H is the Hamming distance between two binary strings. This approximation is justified by the fact that the set of substitutions S was chosen so that the effect of each mutation would be approximately additive, and the number of substitutions that differentiates x from y equals the Hamming distance between $f(x)$ and $f(y)$. Note that Equation (1) relates the Hamming distance between $f(x)$ and $f(y)$ to the binding affinities both of x to y^* and of y to x^*, which is ensured by the symmetry requirement.

In this way, we model the binding affinity of any sequence of D with any sequence of D^*. In principle, the number of sequences may be very large (e.g., later we find sets S of 9 substitutions within 25-nt sequences, which generate sets D of $2^9 = 512$ sequences). However, the distribution of binding affinities is constrained by the corresponding distribution of Hamming distances, so only a subset of these sequences would likely be used for a particular application. Note that these sequences contrast with those generated via random mutations by Nikitin [18] for their designs, because we constrain our substitutions to enable an additive model. While Nikitin observed that NUPACK predictions were not accurate enough to avoid significant experimental debugging, we posit that the highly constrained nature of our substitutions may improve the reliability of the non-orthogonal designs. This claim is supported by our experimental results, although further validation is warranted.

2.2 Design process with isometric graph embeddings

This model establishes a relationship between the binding affinities between D and D^* and the Hamming distances between binary strings. However, it does not solve the problem of designing the sequences themselves given desired binding behavior. To accomplish this, we proposed designing the binary strings for a system directly, and afterwards transforming these into the sequences themselves. This latter step is trivial given binary strings, an initial sequence, and an associated set of substitutions.

A specification of the binding affinities between pairs of sequences can be represented in a matrix $A = (a_{ij})$, where a_{ij} represents the binding affinity between sequences x_i and x_j^* in some units. In analogy to Equation (1), our goal will be to design the differential binding affinities $a_{ij} - a_{i'j'}$ such that

$$a_{ij} - a_{i'j'} \propto \Delta G(x_i, x_j^*) - \Delta G(x_{i'}, x_{j'}^*). \tag{2}$$

To ease design, we chose to consider those design matrices that may be represented in an undirected graph G such that there is some set of vertices $u_i \in V(G)$ where $V(G)$ denotes the vertex set of G. Specifically, we require the existence of a graph such that

$$a_{ij} - a_{i'j'} \propto d_G(u_i, u_j) - d_G(u_{i'}, u_{j'}) \tag{3}$$

where d_G is the shortest path metric on pairs of vertices of G. From this equation, we may immediately conclude the following:

$$a_{ij} - a_{ji} \propto d_G(u_i, u_j) - d_G(u_j, u_i) = 0 \implies a_{ij} = a_{ji} \tag{4}$$

$$a_{ii} - a_{jj} \propto d_G(u_i, u_i) - d_G(u_j, u_j) = 0 \implies a_{ii} = a_{jj} \tag{5}$$

Thus, representation in an undirected graph implies that the design matrix must be symmetric with all diagonal entries identical. In addition, because the shortest path metric is a metric space, the design matrix must also represent a metric space (i.e., satisfy the triangle inequality).

Next, we proposed using a mapping between graphs called an isometric graph embedding, which is a mapping $\phi : V(G) \to V(G')$ between the vertex sets of two graphs that preserves the distances between vertices. That is, for all $u, v \in V(G)$,

$$d_G(u, v) = d_{G'}(\phi(u), \phi(v)). \tag{6}$$

We are particularly interested in isometric graph embeddings into hypercube graphs, or hypercube embeddings. A hypercube graph of dimension m is a graph of 2^m vertices, with each vertex associated with a binary string and two vertices adjacent if and only if their binary strings differ at a single position. These graphs are of interest to us because they naturally represent the Hamming distance between two binary strings in the graph distance between the corresponding hypercube vertices.

For any graph G, finding a hypercube embedding $h : V(G) \to V(H)$ into m-dimensional hypercube graph H immediately implies an assignment of binary strings to each vertex of the graph such that the distance between vertices equals the Hamming distance between binary strings. Given the results of Section 2.1, this completes our stated design goal, assuming we are able to find an initial sequence and a set of substitutions of size at least m:

$$a_{ij} - a_{i'j'} \propto d_G(u_i, u_j) - d_G(u_{i'}, u_{j'}) \tag{7}$$

$$= d_H(h(u_i), h(u_j)) - d_H(h(u_{i'}), h(u_{j'})) \tag{8}$$

$$= \delta_{\text{mut}} \times [\Delta G(x_i, x_j^*) - \Delta G(x_{i'}, x_{j'}^*)] \tag{9}$$

where in the final equality, x_i is the sequence associated with binary string $h(u_i)$, so $f(x_i) = h(u_i)$.

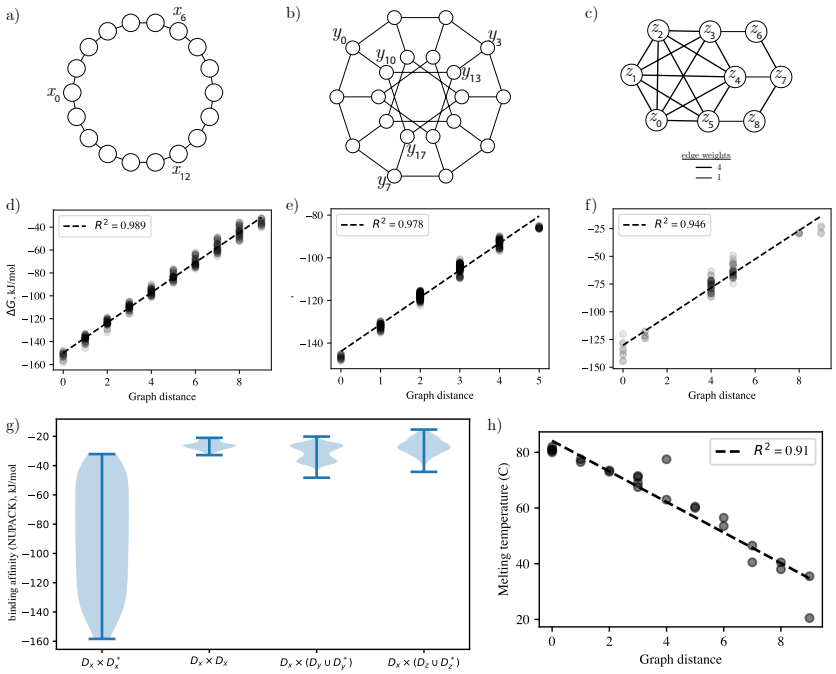


Figure 2 a) A cyclic graph on 18 nodes. b) The Desargues graph. c) A hypercube embeddable weighted graph. d-f) NUPACK simulations of designed sequences show a strong linear relationship between the graph distance and binding affinities of every pair of sequences from D and D^*. g) Distributions of binding affinities between D_x and other sets of sequences. Binding affinites for sequences that should not interact are unfavorable, comparable to the weakest binding affinites observed among the correct pairs of sequences. h) Experimental testing of a subset of the sequences in D_x and D_x^* (Technical Appendix) shows a linear relationship between melting temperature and graph distance. Melting temperature is used here as a proxy for binding affinity.

Given a graph G, the difficulty of finding a hypercube embedding varies depending on the properties of G. For unweighted, undirected graphs, hypercube embeddings were first characterized by Djoković [8], with several important results later established by Winkler and Graham [11, 28]. If an unweighted, undirected graph G permits a hypercube embedding it is called a partial cube, and determining whether a particular graph is a partial cube may be performed in $O(v^2)$ time [9], for a graph with v nodes. In addition, all hypercube embeddings of G are equivalent up to symmetries of the hypercube graph [28]. While not all graphs are partial cubes, many important classes of graphs do permit hypercube embeddings. Two examples are a cyclic graph of $2m$, which is embeddable into an m-dimensional hypercube, and the Desargues graph on 20 nodes, which is embeddable into a 5-dimensional hypercube (Figure 2ab).

To give an intuition for how such embeddings are constructed, we briefly describe the algorithm proposed by Graham and Winkler [11], noting that more efficient algorithms exist. First, a relation θ is defined on the edge set $E(G)$ of the graph G, where $uv\theta u'v'$ if and only if the quantity $[d(u,u') - d(u,v')] - [d(v,u') - d(v,v')]$ is nonzero. The transitive closure $\hat{\theta}$ of θ is an equivalence relation, and the equivalence classes of $\hat{\theta}$ partition $E(G)$ into sets $E_1, \ldots, E_m$. For set E_i, let G_i be the graph with $V(G_i) = V(G)$ and $E(G_i) = E(G) \setminus E_i$ (i.e., formed by removing the edges in E_i from G). If every G_i has exactly two connected components, then G is hypercube embeddable. In this case, a hypercube embedding may be constructed by assigning binary strings to each vertex of G such that the i-th bit of a vertex is 0 if it is one connected component of G_i and 1 if it is in the other.

When a graph G is weighted, the task of finding a hypercube embedding is NP-hard [7], so an efficient and generally-applicable solution to the problem is unlikely to exist. However, methods for efficiently embedding some classes of weighted G exist (Figure 2c) [5, 24, 25]. Characterizing additional classes of weighted graphs for which hypercube embeddings can be efficiently found remains an open area of research in graph theory, and advances in this area would improve the utility of this design process.

2.3 Validation via simulation and experiment

We consider three graphs of varying complexity (Figure 2a-c), and construct hypercube embeddings and sequence designs for each. For the three designs, we denote the designed sequences as D_x, D_y, and D_z, and the final binary strings and sequences are given in the Technical Appendix. To show the robustness of our design, each design used a distinct initial sequence taken from a previously designed set of 240,000 25-nt PCR primers known to be orthogonal to one another [29]. In each case, the pairwise binding affinities between the corresponding set of sequences D_i and the set of complementary sequences D_i^*, $i \in \{x, y, z\}$ were simulated with NUPACK at 25 °C in an aqueous solution of 1M NaCl. The simulations show a strong linear correlation between the binding affinities and the original graph distances (Figure 2d-f), with correlation coefficients of $R^2 = 0.99$, 0.98, and 0.95, respectively for the three designs.

To test the orthogonality of separately designed sets of non-orthogonal sequences, we computed binding affinities between sets of sequences that should not interact. Specifically, we tested the set D_x against the sets D_x, D_y, D_y^*, D_z, and D_z^*, and found weak binding affinities in all cases (Figure 2g). The binding affinities were comparable with the weakest binding observed among any of the expected pairs of interacting sequences.

Experimental tests of the first sequence design (D_x and D_x^*) support the validity of our model. As a proxy for binding affinity, we measured melting temperatures for 25 pairs of sequences (5 sequences of D_x and 5 sequences of D_x^*), covering a range of graph distances. Melting temperature was interpreted as a proxy for binding affinity, with higher melting temperature indicating stronger binding affinity. Melting temperatures plotted against the expected distance in the cyclic graph showed a strong linear correlation with a correlation coefficient of $R^2 = 0.91$ (Figure 2h).

3 Analog computation with promiscuous ligand-receptor networks

In naturally occurring biological networks, promiscuous binding between components (non-orthogonality) is common [1, 15]. Recently, researchers studied the computational abilities of promiscuous ligand-receptor networks, such as the BMP signaling pathway [1]. In their model of this pathway, trimers are formed from the binding of a ligand to two receptor halves, with each trimer having its associated free energy and signal output production rate. The ligand concentrations, which were assumed to be in excess, affect the formation of trimers and subsequent signal output, and these concentrations were the network inputs. The receptor concentrations, binding affinities, and trimer production rates constitute network parameters that determine the particular function computed by the system. The authors identified four archetypal response types characteristic of the promiscuous ligand-receptor network architecture, which they labeled “additive”, “ratiometric”, “balance”, and “imbalance” gates [1]. In this section, we propose and simulate a DNA strand displacement analogue to the promiscuous ligand-receptor network.

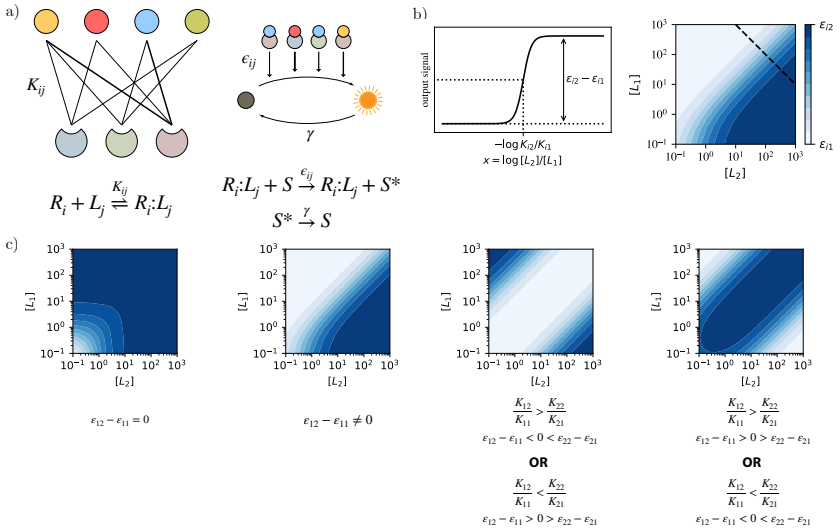


Figure 3 a) The dimer model for a promiscuous ligand-receptor network uses a set of K_{ij} binding affinities between each pair of ligands and receptors, and a set of ϵ_{ij} reaction rates to represent the signal activation rates associated with each ligand-receptor pair. b) The signal response of each receptor is independent of the other receptors, because we assume high ligand concentrations. For a two-ligand network, the response is sigmoidal in the log-ratio of the two ligand concentrations (left). This is also seen in a diagonal cross-section (dotted line) of a log-log heatmap (right). c) From left to right, an additive response, a ratiometric response, an imbalance response, and a balance response. Below each heatmap are conditions on K_{ij} and ε_{ij} to achieve each response.

3.1 Dimer model for the promiscuous ligand-receptor network

Here, we present a simplified model of a promiscuous ligand-receptor network, based only on dimer formation between ligands and receptors. While less biologically pertinent, the dimer model is capable of computing the same four archetypal response types. The dimer model considers a system of n_R receptors and n_L ligands, such that each ligand-receptor pair may bind and generate signal output with an arbitrary binding affinity and signal production rate (Figure 3a). For simplicity, we let $\varepsilon_{ij} = \frac{\epsilon_{ij}}{\gamma}$. Under the limit of high total ligand concentration, we may assume that all receptors are bound to some ligand, and the steady-state network output is given by

$$[S]_{\text{ss}} = \sum_{1 \leq i \leq n_R} [R_i]_0 \frac{\sum_{1 \leq j \leq n_L} \varepsilon_{ij} K_{ij} [L_j]_0}{\sum_{1 \leq j \leq n_L} K_{ij} [L_j]_0}, \tag{10}$$

where $[A]_0$ is the total concentration of species A, bound or unbound. A full mathematical analysis of the model is given in the Technical Appendix.

In our case, we are interested in a system of two ligands and two receptors, which is already capable of generating the four archetypal response types identified in the context of the BMP signaling pathway [1]. We consider the system response as a function of $x = \log \frac{[L_2]}{[L_1]}$, because in the limit of high ligand concentration, the signal output of a two-ligand system is a function only of the ratio of the two ligand concentrations. Consider the receptor R_1. As the x increases (L_2 dominates L_1), the receptor transitions from binding predominantly to L_1 to binding predominantly to L_2. This transition follows a sigmoidal function of x, and the signal output due to R_1 is a sigmoid whose minimum and maximum are determined by the output production rates of $R_1{:}L_1$ and $R_1{:}L_2$ (Figure 3b). Because ligand concentration is high, the binding of R_2 to L_1 and L_2 is not affected by that of R_1, so the total network output is a sum of the two sigmoidal responses generated by the two receptors.

Each of the four archetypal response types is generated from a linear combination of two sigmoids, and different responses may be chosen by transforming the sigmoidal response of each receptor. The sigmoidal response of a receptor can be shifted horizontally by adjusting the relative binding affinity of the receptor to each ligand. The response can be shifted and scaled vertically by adjusting the output production rates of its dimers. Parameter constraints and dimer model simulations for each response type are shown in Figure 3c. Note that the additive and ratiometric responses require only a single receptor.

3.2 DNA strand displacement-based implementation

We designed a set of DNA-based "ligands" and "receptors" to implement the behavior of the dimer model for a promiscuous ligand-receptor network. Each DNA-based ligand and receptor uses a combination of classical orthogonal domains along with additional domains taken from two sets of independently designed non-orthogonal sequences (Figure 4a). The signal output is generated by a fluorophore-quencher FRET pair, so that the fluorophore is quenched in certain ligand-receptor conformations. Changes in response type are possible simply by swapping out the choice of non-orthogonal domains on each DNA-based ligand and receptor.

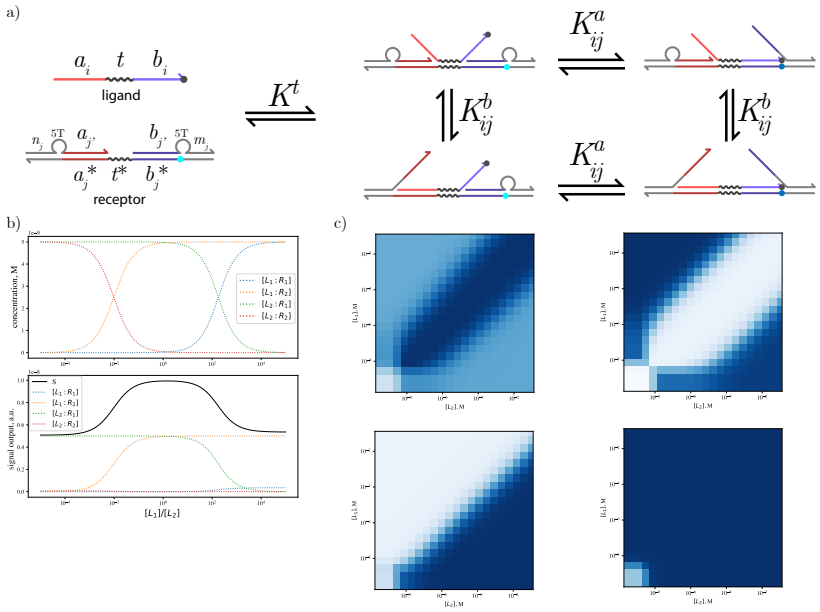


Figure 4 a) A proposed DNA strand displacement implementation of a promiscuous ligand-receptor network. b) For a balance gate, NUPACK simulations of the concentrations of the four ligand-receptor pairs as well as the signal response from each. The signal response is estimated from the probability that the 3′-most nucleotide of the ligand is bound to its corresponding nucleotide on the bottom strand of the receptor. c) Heatmaps from NUPACK simulations of our sequence designs for each of the four archetypal response types.

The steady-state output of this network is given by the following equation

$$\sum_{1\leq i\leq n_R, 1\leq j\leq n_L} [(R_i{:}L_j)^*] = \sum_{1\leq i\leq n_R} \frac{[R_i]_0 \sum_{j=1}^{n_L} K^t(1+K^a_{ij})[L_j]_0}{\sum_{j=1}^{n_L} K^t(1+K^a_{ij})(1+K^b_{ij})[L_j]_0}, \tag{11}$$

where $[(R_i{:}L_j)^*]$ is the concentration of the ligand-receptor pair in a fluorescent (i.e., not quenched) state (see the Technical Appendix for a full derivation). This equation can be made formally equivalent to Equation (10) via the following substitutions

$$K_{ij} = K^t(1 + K^a_{ij})(1 + K^b_{ij}) \tag{12}$$

$$\varepsilon_{ij} = \frac{1}{1 + K^b_{ij}} \tag{13}$$

which relate the parameters of this DNA strand displacement system (K^t, K^a_{ij}, and K^b_{ij}) to the parameters of the dimer model of the previous section (K_{ij} and ε_{ij}). In this way, it is possible in principle to implement a wide variety of dimer model parameters, limited mainly by a discretization of the energy landscape due to δ_{mut} (the change in binding affinity due to a single substitution mutation).

We describe our implementation of the balance gate in detail. Implementation of the other three gates follows similarly. For our design, we used the binary strings and corresponding sequences from the first two designs in Section 2.3, D_x and D_y. To achieve the network parameters for the balance gate (Figure 3c), we chose 2 binary strings from the D_x design and 4 binary strings from the D_y set of sequences. Sequences corresponding to each binary string were designed, and NUPACK simulation was used to estimate K_{ij} and ε_{ij} using these particular sequences and Equations (12) and (13), with K^t set to 1 as it did not affect the relative values of the K_{ij}) (Table 1). Complete lists of sequences are given in the Technical Appendix.

Table 1 Table of estimated K_{ij} and ε_{ij} values for the balance gate.

K_{ij}	L_1	L_2
R_1	4.0	1.5×10^5
R_2	1.8×10^4	4.0

ε_{ij}	L_1	L_2
R_1	0.5	1.0
R_2	1.0	0.5

NUPACK does not allow direct simulation of the fluorophore or quencher modifications, so the concentration of colocalized fluorophore-quencher pairs was estimated from the probability that the rightmost nucleotide of the y domain on a ligand would be bound to its corresponding nucleotide on a receptor. Note that hydrophobic interactions in an actual experiment could stabilize this conformation further. Using NUPACK, we simulated the concentration of each ligand-receptor pair for a fixed total ligand concentration $[L_1]_0 + [L_2]_0 = 2$ μM and varying ligand concentration ratio (Figure 4b, top). This data shows the two transitions corresponding to replacement of L_1 by L_2 on each receptor. The computed network output was also computed, as well as the output from each ligand-receptor complex, showing the expected balance gate behavior (Figure 4b, bottom). Finally, a heatmap was generated showing the network output with initial receptor concentrations $[R_1]_0 = [R_2]_0 = 5$ nM and ligand concentrations ranging from 10^{-9} to 10^{-1} M. This heatmap shows that at high ligand concentration the signal output is high when the ligand concentrations are similar and low otherwise. Note that at low ligand concentrations, receptors remain unbound and the signal output will be low regardless of the ligand concentration ratio.

For the other three archetypal response types, the binary strings were selected based on the constraints for each response type in Figure 3c, and DNA sequences and response heatmaps were computed using NUPACK simulation (Figure 4c). Results show successful implementation of each response type. Note that the additive and ratiometric gates required only a single receptor for their implementations because their response profile consisted of a single sigmoid.

4 Digital logic computation with non-orthogonal DNA hairpins

Next, we wished to demonstrate the utility of our design process for digital logic computation. We were motivated by recent work due to Nikitin illustrating digital logic gates based on the binding of short ssDNA strands optimized for particular binding affinities [18]. Notably, Nikitin used sequences that were freely mutated and subsequently tested with NUPACK to achieve particular binding affinities, and additional optimization of the affinities and strand concentrations was needed for successful circuit function. The author also found that NUPACK predictions for sequences mutated in this manner were not accurate enough to avoid significant experimental trial and error [18]. This contrasts with our approach of carefully selecting the locations of each mutation and the nucleotide identities in order to achieve consistent and predictable effects on binding affinity within the population of sequences of our design. In addition, strand concentrations were largely determined from the circuit connectivity, with only a single global working concentration that required optimization.

In Section 4.1, we begin by describing the basic structure of these logic gates, which is based off of a fan-in design for a NOR gate, and our hairpin-based implementation, which we call a "hairpin logic circuit". Section 4.2 describes the application of our design process to logic circuits of varying complexity. This process is based on the construction of an interaction graph for which a hypercube embedding may be found as in Section 2.2.

4.1 Structure of the hairpin logic circuit

The gates proposed by Nikitin [18] use sequences of NOR gates implemented by controlled binding between single strands of DNA. For each NOR gate, any one of a set of input strands may bind to the output strand, preventing it from binding to any downstream strands. In contrast to Nikitin's proposed design, our sequence designs are intended for control of the differential binding affinities of distinct double-stranded DNA complexes, rather than the change in free energy associated with two single strands forming a duplex. Thus, we proposed a hairpin-based circuit design in which each hairpin stem consists of two non-orthogonal domains that may either bind to each other (stem closed) or to another hairpin (stem open) (Figure 5a). For hairpin signal X, we denote the non-orthogonal stem domains with lowercase x and $\bar{x}^*$. The structure of our circuits is identical to those of Nikitin, except for the substitution of each single-stranded signal molecule with one of our hairpin molecules. For example, the NOT gate is simply a single-input NOR gate (Figure 5b), whose output signal is high when the input is low using a FRET pair to modulate fluorescence. Other examples of simple gates constructed from the NOR-gate primitive are shown in Figure 5c.

The favorability of two hairpins A and B opening and binding to each other is dependent on several factors. These include the degree of sequence complementarity in their stems, the favorability of opening up the hairpin loop, and the entropic penalty of replacing two freely moving molecules with a single molecule. Our sequence design process controls the first factor (i.e., the affinity of a to $\bar{b}^*$ and b to $\bar{a}^*$ relative to the affinity of a to $\bar{a}^*$ and b to $\bar{b}^*$). The latter two factors are dependent on system parameters, such as the strand concentrations and the design of the hairpin loop, and are expected to be approximately equal for all hairpin signal binding reactions.

For each circuit, we assume a working concentration c, which can be increased or decreased to tune the global favorability of a free (closed) hairpin state against the sequestered (open) hairpin state in which it is bound to another (open) hairpin signal. Because some hairpin signals must bind to multiple downstream signals as part of their function, the concentration of each signal was set to the total concentration of the downstream signals to which it must

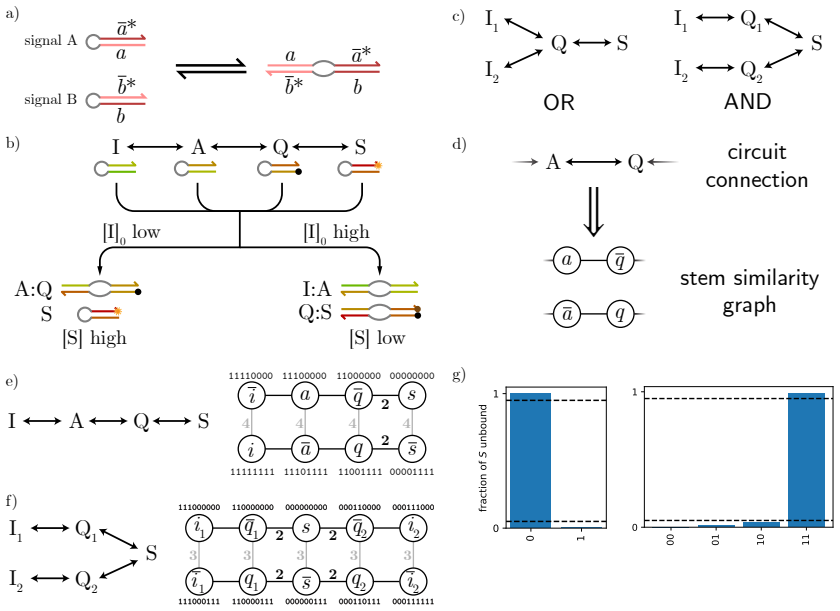


Figure 5 a) The hairpin logic circuit uses signal strands that form either a closed hairpin when isolated or an open conformation when bound to another signal. b) Example operation of a NOT gate. Each Q hairpin has a quencher label that reduces fluorescence from S when they are bound. c) Each layer of the circuit is a NOR gate, allowing the implementation of any other logic gate. Circuits for an OR and an AND gate are shown. d) To design the domains for each hairpin stem, a stem similarity graph is constructed that represents the desired binding relationships between each pair of stem domains. Each circuit connection corresponds to two edges in the stem similarity graph, to ensure that the domains on the two hairpins can bind to each other. e-f) Stem similarity graphs and hypercube embeddings for the NOT and AND gates. g) NUPACK simulations of circuit output for each gate. Performance is within 5% of the expected output (dotted lines) for all inputs.

bind. In principle, this means some hairpins will be at higher concentration and thus more likely to bind to other hairpins; however, in practice, this did not materially affect the circuit behavior (free energy of binding is affected by the logarithm of concentration, so orders of magnitude change would likely be required to affect the circuit).

4.2 Sequence design for the hairpin logic circuit

From the connectivity of a circuit diagram, we can graphically represent the desired similarity between hairpin stem domains. For a circuit C, the hairpin stem similarity graph $\mathcal{H}(C)$ represents each hairpin stem domain with a vertex (i.e., two vertices per hairpin signal). If two hairpin signals A and B are connected in the circuit diagram, then $a\bar{b}$ and $\bar{a}b$ are (unweighted, undirected) edges in $\mathcal{H}(C)$ (Figure 5d). This implies that a and $\bar{b}$ should differ by only a single mutation, and similarly for $\bar{a}$ and a. The stem similarity graph for a NOT gate is shown in Figure 5e. In this simple case, the stem similarity graph has two connected components, which means that the graph distance for the domains in each hairpin stem can be freely chosen (i.e., a to $\bar{a}$ for hairpin signal A). For the NOT gate, we assign a graph distance of 4 between every pair of domains on a single hairpin, which corresponds to 4 mutations within each hairpin stem. The implications of this choice are discussed further later.

When a circuit has at least 3 layers, it becomes necessary to adjust the stem similarity graph so that the output signals bind less strongly to their connected signals. This ensures that the output signal will only bind to any preceding signals if those signals have no other binding partners. To accomplish this with the AND gate, we apply a weight of 2 to each edge incident to a non-orthogonal domain of the output hairpin signal (Figure 5f).

Given a stem similarity graph, a hypercube embedding can be found in $O(|V|^2)$ time if one exists, for a graph with $|V|$ vertices. Hypercube embeddings for the NOT and AND gates are shown in Figure 5ef. A unitless binding affinity of two hairpin signals, neglecting factors other than stem loop complementarity, can be estimated with the equation

$$\delta(A, B) = d(a, \bar{b}) + d(b, \bar{a}) - d(a, \bar{a}) - d(b, \bar{b}) \tag{14}$$

Note that two hairpins that are intended to bind are guaranteed will always have the lowest binding affinities because each stem domain was separated by only a single edge in the stem similarity graph. This is true even for the output signal domains when they have incident edge weights of 2.

Corresponding sequences for the NOT and AND gate designs are taken from the sequence set D_y (Section 2.3; see the Technical Appendix for full set of sequences). When fewer than 9 substitutions were needed, such as with the NOT gate, a subset of the 9 substitutions available with this design was selected. In addition, the order of the substitutions was randomized, in order to distribute the mutations more evenly across the sequence. This randomization was performed for the other gates as well. Using NUPACK, the circuit output for high and low initial input concentration was computed as the concentration of the closed hairpin S (Figure 5g), showing performance within 5% of the optimal values.

To ensure that only desired hairpin pairs would bind, we modulated the working concentration for each circuit so that binding between undesired hairpin pairs was less favorable than the hairpins remaining free (closed). Previously, we chose a graph distances of 4 (NOT gate) or 3 (AND gate) between pairs of domains corresponding to the same hairpin signal, to demonstrate the effect of this design choice. With the NOT gate, a working concentration of 10^{-6} M was most effective, while with the AND gate, a significantly higher working concentration of 10^{-4} M was used. In general, this graph distance can be increased at the cost of additional bits in the hypercube embedding, in order to allow a lower working concentration to correctly balance the open and closed hairpin states. This flexibility can be useful, for instance, if the working concentration is fixed by other experimental factors and cannot be independently modulated.

For some circuits, the stem similarity graph constrains the graph distance between the two domains on each hairpin signal. The simplest feedforward circuit for which this occurs is the logically false circuit for $\mathrm{NOR}(A, \neg A)$, which can be implemented with 3 signals A, $B = \neg A$, and $C = \mathrm{NOR}(A, \neg A)$ (Figure 6a). In this case, the corresponding stem similarity graph constrains the distance between a and $\bar{a}$ to be 3 (similarly for B and C). Situations such as this occur whenever the circuit is not bipartite, when viewed as a graph on the hairpin signals themselves. This becomes increasingly common with larger circuits, and in such cases, the working concentration must be chosen based on the constraints implied by the stem similarity graph.

As a final example, we consider the square root circuit first considered by Qian and Winfree [20], which was also implemented by Nikitin [18] (Figure 6b). The stem similarity graph for this circuit is significantly more complex than the previous examples (Figure 6c), consisting of 30 vertices and 34 edges. As with the AND gate, a weight of 2 was assigned to the edges incident to the domains of the output signals S_1 and S_2. This stem similarity graph

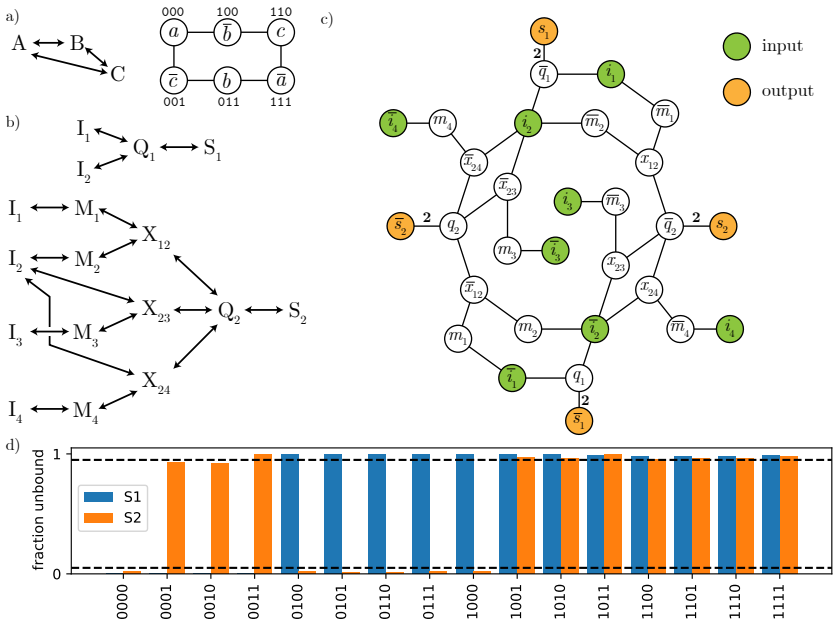


Figure 6 a) When a stem similarity graph is connected, the graph distance between domains on the same hairpin is constrained. This is a circuit for A NOR NOT A, which is logically false. b) Circuit connections for the square-root circuit. This is the same circuit that was constructed by Nikitin [18]. c) Stem similarity graph for the square root circuit. Domains on the input and output hairpins are colored for clarity. d) NUPACK simulations of circuit output for all 16 input combinations (inputs are listed in the order $I_1I_2I_3I_4$). Outputs remain within 10% of the correct values for all inputs, and within 5% (dotted lines) for all but 2 of the inputs (0001 and 0011).

is not exactly hypercube embeddable. For this case, we introduce the notion of a *k-hypercube embedding*, or an isometric embedding into a hypercube that preserves all distances up to a distance of k. Precisely, $\phi : V(G) \to V(H)$ is a k-hypercube embedding if for all $u, v \in V(G)$:

1. $d_G(u, v) = d_H(\phi(u), \phi(v))$ if $d_G(u, v) \leq k$
2. $d_G(u, v) > k$ if $d_G(u, v) > k$

This notion is a generalization of the k-snake, which is a k-hypercube embedding of a linear graph [13, 26].

The use of a k-hypercube embedding has an additional advantage, in that it may allow the researcher to embed into a hypercube of smaller dimension (i.e., requiring fewer bits). Unfortunately, an efficient algorithm for finding a k-hypercube embedding is not known, although for smaller circuits this is often possible to do by hand.

For the square root circuit, we constructed a 2-hypercube embedding of dimension 9, where $k = 2$ was chosen because all correct interactions were encoded in the stem similarity graph with a distance of at most 2 (see Technical Appendix). Any vertices at distance 3 or greater should not interact anyway, and the working concentration will be chosen to ensure that these interactions are not favorable. Our 2-hypercube embedding has the added advantage that for every hairpin, its two domains have the same Hamming distance from each other. This makes it easier to choose a working concentration that correctly tunes the propensity of each hairpin signal to choose an open conformation (bound to another hairpin) over a closed one.

To simulate this circuit, sequences corresponding to each binary string were used from the set D_y from Section 2.3. A working concentration of 10^{-11} M was used; all signals were initially present at this concentration except for I_1 and I_2, which used concentrations of

2×10^{-11} and 4×10^{-11}, respectively. NUPACK simulations were performed over all 16 possible 4-bit inputs, which showed that, for all input combinations, the 2-bit output was within 10% of the correct value (Figure 6d). All but two inputs (0001 and 0011) achieved outputs within 5% of the correct values. It is possible that additional circuit optimization could improve this further.

We make one final comments regarding this application of our sequence design procedure to digital logic circuits. When some hairpin signals have stems that bind more strongly than others, choosing a global working concentration can be difficult. In the square root circuit, we circumvented this problem in the way we compressed our hypercube embedding; however, this may not always be possible. Handling this scenario remains another open question that could make it possible to apply our design process to additional circuits.

5 Discussion and Conclusions

In this work, we have presented an effective rational design approach for non-orthogonal DNA domains, that may be combined with orthogonal domains for complex DNA strand displacement circuitry. We applied this approach to two previously proposed molecular computational systems based on non-orthogonal interactions, through which we demonstrated *in silico* analog and digital logic computation. This design approach eases the use of non-orthogonal domains within DNA strand displacement cascades, because it allows researchers to perform initial designs of non-orthogonal domains using only binary strings. Experimental implementation requires finding an appropriate starting DNA sequence with enough substitutions; heuristically, a starting sequence with little secondary structure and that binds to its perfect complement primarily in a single conformation works well. Once found, generating the corresponding DNA sequence variants is straightforward, allowing researchers to quickly design candidate non-orthogonal domains. Thus, a larger portion of the design process can be done prior to ordering materials. Further experimental testing is warranted to quantify the accuracy of our design method.

Several areas for further improvements to our method exist. For example, the use of k-hypercube embeddings was necessary for the more complex square root circuit. Additional work could address algorithms for efficiently constructing k-hypercube embeddings, which would also help with compressing the embeddings so they can be implemented on shorter DNA strands.

The sequence design process we describe here has the potential to be applied in other contexts. For example, previous DNA strand displacement-based implementations of neural networks have used "weight complexes," dedicated auxiliary DNA molecules whose concentrations encode the weights between various nodes [6, 21]. However, as the size of a neural network grows, the number of weights that must be encoded grows quadratically in the number of nodes, so that the number of distinct DNA molecules that must be designed and synthesized grows quickly. Using non-orthogonal sequence design, it may be possible to encode weights in the binding affinities between molecules, which would significantly reduce the number of system components required, increasing the size of the neural networks that can feasibly be implemented experimentally.

Recently, researchers have shown that non-orthogonal interactions can also be applied to nanostructure fabrication, allowing the design of a pool of structural subunits capable of creating any one of several possible multi-subunit assemblies. The creation of a particular assembly can be triggered by the presence of a nucleation seed [16] or by the concentrations of the various subunits [30]. An experimental demonstration of this used a set of 917 unique

DNA tiles to construct one or more of three target assemblies [10]. This demonstration used orthogonal interactions between tiles; however, our sequence design method could in principle be used to allow the construction of a larger number of target assemblies closer to the theoretical limit [16].

The problem of non-orthogonal sequence design is a complex task, with different approaches likely to be best suited to different applications. However, there are many potential use cases for a non-orthogonal sequence design approach that can be applied to a variety of DNA strand displacement systems. We hope that our non-orthogonal sequence design method spurs new innovation in both DNA sequence design and the computational uses of non-orthogonality, and that future improvements to non-orthogonal design techniques will open the doors to more complex DNA-based computers and new advancements in molecular computing.

References

1 Yaron E Antebi, James M Linton, Heidi Klumpe, Bogdan Bintu, Mengsha Gong, Christina Su, Reed McCardell, and Michael B Elowitz. Combinatorial signal perception in the BMP pathway. *Cell*, 170(6):1184–1196, 2017.

2 Eric B Baum. Building an associative memory vastly larger than the brain. *Science*, 268(5210):583–585, 1995.

3 Callista Bee, Yuan-Jyue Chen, Melissa Queen, David Ward, Xiaomeng Liu, Lee Organick, Georg Seelig, Karin Strauss, and Luis Ceze. Molecular-level similarity search brings computing to DNA data storage. *Nature communications*, 12(1):1–9, 2021.

4 Joseph Berleant. *DNA sequence design of non-orthogonal binding networks, and application to DNA data storage.* PhD thesis, Massachusetts Institute of Technology, 2023.

5 Joseph Berleant, Kristin Sheridan, Anne Condon, Virginia Vassilevska Williams, and Mark Bathe. Isometric Hamming embeddings of weighted graphs. *Discrete Applied Mathematics*, 332:119–128, 2023.

6 Kevin M Cherry and Lulu Qian. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, 559(7714):370–376, 2018.

7 Vašek Chvátal. Recognizing intersection patterns. *Annals of Discrete Mathematics*, 8:249–252, 1980. doi:10.1002/net.3230210602.

8 Dragomir Ž Djoković. Distance-preserving subgraphs of hypercubes. *Journal of Combinatorial Theory, Series B*, 14(3):263–267, 1973.

9 David Eppstein. Recognizing partial cubes in quadratic time. *Journal of Graph Algorithms and Applications*, 15(2):269–293, 2011.

10 Constantine Glen Evans, Jackson O'Brien, Erik Winfree, and Arvind Murugan. Pattern recognition in the nucleation kinetics of non-equilibrium self-assembly. *arXiv preprint arXiv:2207.06399*, 2022.

11 Ronald L Graham and Peter M Winkler. On isometric embeddings of graphs. *Transactions of the American Mathematical Society*, 288(2):527–536, 1985.

12 Haukur Gudnason, Martin Dufva, Dang Duong Bang, and Anders Wolff. Comparison of multiple DNA dyes for real-time pcr: effects of dye concentration and sequence composition on DNA amplification and melting temperature. *Nucleic acids research*, 35(19):e127, 2007.

13 Simon Hood, Daniel Recoskie, Joe Sawada, and Dennis Wong. Snakes, coils, and single-track circuit codes with spread k. *Journal of Combinatorial Optimization*, 30:42–62, 2015.

14 Matthew R Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. Visual dsd: a design and analysis tool for dna strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.

15 Tomas Malinauskas and E Yvonne Jones. Extracellular modulators of Wnt signalling. *Current opinion in structural biology*, 29:77–84, 2014.

16 Arvind Murugan, Zorana Zeravcic, Michael P Brenner, and Stanislas Leibler. Multifarious assembly mixtures: Systems allowing retrieval of diverse stored structures. *Proceedings of the National Academy of Sciences*, 112(1):54–59, 2015.

17 Andrew Neel and Max Garzon. Semantic retrieval in DNA-based memories with Gibbs energy models. *Biotechnology progress*, 22(1):86–90, 2006.

18 Maxim P Nikitin. Non-complementary strand commutation as a fundamental alternative for information processing by DNA and gene regulation. *Nature Chemistry*, pages 1–13, 2023.

19 Nicolas Peyret, P Ananda Seneviratne, Hatim T Allawi, and John SantaLucia. Nearest-neighbor thermodynamics and NMR of DNA sequences with internal A⊙A, C⊙C, G⊙G, and T⊙T mismatches. *Biochemistry*, 38(12):3468–3477, 1999.

20 Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

21 Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475:368–72, July 2011. doi:10.1038/nature10262.

22 John SantaLucia Jr. and Donald Hicks. The thermodynamics of DNA structural motifs. *Annual Review of Biophysics and Biomolecular Structure*, 33(1):415–440, 2004. doi:10.1146/annurev.biophys.32.110601.141800.

23 Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-free nucleic acid logic circuits. *science*, 314(5805):1585–1588, 2006.

24 Kristin Sheridan, Joseph Berleant, Mark Bathe, Anne Condon, and Virginia Vassilevska Williams. Factorization and pseudofactorization of weighted graphs. *Discrete Applied Mathematics*, 337:81–105, 2023. doi:10.1016/j.dam.2023.04.019.

25 Sergey V. Shpectorov. On scale embeddings of graphs into hypercubes. *Eur. J. Comb.*, 14(2):117–130, March 1993. doi:10.1006/eujc.1993.1016.

26 Richard C Singleton. Generalized snake-in-the-box codes. *IEEE Transactions on Electronic Computers*, pages 596–602, 1966.

27 Kyle J Tomek, Kevin Volkel, Elaine W Indermaur, James M Tuck, and Albert J Keung. Promiscuous molecules for smarter file operations in DNA-based data storage. *Nature Communications*, 12(1):3518, 2021.

28 Peter M Winkler. Isometric embedding in products of complete graphs. *Discrete Applied Mathematics*, 7(2):221–225, 1984.

29 Qikai Xu, Michael R Schlabach, Gregory J Hannon, and Stephen J Elledge. Design of 240,000 orthogonal 25mer DNA barcode probes. *Proceedings of the National Academy of Sciences*, 106(7):2289–2294, 2009.

30 Weishun Zhong, David J Schwab, and Arvind Murugan. Associative pattern recognition through macro-molecular self-assembly. *Journal of Statistical Physics*, 167:806–826, 2017.

A Methods

A.1 NUPACK simulation parameters

All NUPACK simulations were performed at 25 °C, with 1 M NaCl. NUPACK simulations for Sections 2 and 3 used material "dna04-nupack3" and ensemble "some-nupack3". NUPACK simulations for Section 4 used material "dna" and ensemble "stacking".

A.2 DNA melting assay

The sequences x_0, x_1, x_3, x_6, and x_{10} and their complements were tested. Because intercalating dyes such as SYBR Green often affect the binding affinity of DNA strands or have strong variations in fluorescence with temperature [12], we used a FRET assay with strands labeled either with cyanine 3 (Cy3) or cyanine 5 (Cy5) dyes. Cy3- or Cy5-labeled strands were ordered from Integrated DNA Technologies (IDT). Strands were mixed to a concentration

of 0.25 μM per strand in a solution of 1× TAE and 1 M NaCl. Fluorescence of Cy3 was measured on a qPCR machine (QuantStudio Flex 6, Applied Biosystems) over a temperature ramp from 95 °C down to 10 °C and then back to 95 °C, at a rate of 0.5 °C/min. This was repeated three times. Each curve was corrected with the Cy3 fluorescence measured from a sample of the single strand without any binding partner. Data from the upward ramp was averaged, and the average rate of change in fluorescence with temperature was computed. The temperature with highest rate of change was taken as the melting temperature.

B Thermodynamic analysis of the promiscuous ligand-receptor system

B.1 Analysis of the dimer model

The dimer model uses a set of n_R receptors R_i and n_L ligands L_j and is capable of generating the four archetypal response types. This requires fewer components than the trimer model used in by Antebi et al. in their analysis of the BMP signaling pathway [1].

$$R_i + L_j \underset{}{\overset{K_{ij}}{\rightleftharpoons}} D_{ij} \tag{15}$$

$$D_{ij} \xrightarrow{\epsilon_{ijk}} D_{ij} + S \tag{16}$$

$$S \xrightarrow{\gamma} \emptyset \tag{17}$$

Let $\varepsilon_{ij} = \frac{\epsilon_{ij}}{\gamma}$. Given these reactions, the steady-state level of S is

$$[S]_{\text{ss}} = \sum_{1\le i\le n_R, 1\le j\le n_L} \varepsilon_{ij}[D_{ij}]_{\text{ss}} \tag{18}$$

$$= \sum_{1\le i\le n_R} [R_i]_0 \frac{\sum_{1\le j\le n_L} \varepsilon_{ij} K_{ij}[L_j]_{\text{ss}}}{1 + \sum_{1\le j\le n_L} K_{ij}[L_j]_{\text{ss}}} \tag{19}$$

$$\approx \sum_{1\le i\le n_R} [R_i]_0 \frac{\sum_{1\le j\le n_L} \varepsilon_{ij} K_{ij}[L_j]_0}{\sum_{1\le j\le n_L} K_{ij}[L_j]_0} \tag{20}$$

where $[A]_0$ is the total concentration of species A bound or unbound and the final approximation is achieved by assuming the total ligand concentration is high.

When there are exactly two ligands, each term of the summation can be individually expressed as a sigmoid curve:

$$[S]_{\text{ss}} \approx \sum_{1\le i\le n_R} [R_i]_0 \frac{\varepsilon_{i1} K_{i1}[L_1]_0 + \varepsilon_{i2} K_{i2}[L_2]_0}{K_{i1}[L_1]_0 + K_{i2}[L_2]_0} \tag{21}$$

$$= [R_1]_0 \left[\varepsilon_{11} + \frac{\varepsilon_{12} - \varepsilon_{11}}{\frac{K_{11}}{K_{12}} e^x + 1}\right] \tag{22}$$

where $x = \log\left(\frac{[L_1]_0}{[L_2]_0}\right)$. This describes a sigmoid curve that transitions from $[R_i]_0\varepsilon_{i2}$ to $[R_i]_0\varepsilon_{i1}$ with a midpoint at $-\log\frac{K_{i1}}{K_{i2}}$. The "height" of the sigmoid is given by $[R_i]_0(\varepsilon_{i2} - \varepsilon_{i1})$.

Thus, in a two ligand, two receptor system, the signal response is the sum of two sigmoids, each of which may be independently shifted along the x-axis by changing $\frac{K_{i1}}{K_{i2}}$, and shifted and/or stretched along the vertical axis by modifying ε_{i1} and ε_{i2}. Note that the steepness of the sigmoid (i.e. horizontal stretching) is not adjustable under this model; the same is true of the trimer model presented by Antebi et al. [1].

The additive and ratiometric responses may be generated by a single receptor, while the balance and imbalance responses require two receptors.

B.2 DNA strand displacement implementation

Consider the ligands and receptors shown in Figure 4a. Each ligand has two non-orthogonal domains (a_i and b_i) as well as a toehold that allows it to bind to each receptor. Once bound, the two non-orthogonal domains on the receptor that flank its toehold binding site may be displaced with some free energy change based on the relative affinities of the incumbent domains (a_j and b_j) and intruder domains to the bottom domains (a_k and b_k). A FRET-based readout is implemented by the addition of a fluorophore-quencher pair, which is activated only when the right-hand non-orthogonal domain is displaced. The full reaction diagram is enumerated below:

$$R_i + L_j \overset{K^t}{\rightleftharpoons} R_i{:}L_j \tag{23}$$

$$R_i{:}L_j \overset{K^a_{ij}}{\rightleftharpoons} (R_i{:}L_j)_a \tag{24}$$

$$R_i{:}L_j \overset{K^b_{ij}}{\rightleftharpoons} (R_i{:}L_j)_b \tag{25}$$

$$(R_i{:}L_j)_a \overset{K^b_{ij}}{\rightleftharpoons} (R_i{:}L_j)_{ab} \tag{26}$$

$$(R_i{:}L_j)_b \overset{K^a_{ij}}{\rightleftharpoons} (R_i{:}L_j)_{ab} \tag{27}$$

Thermodynamic analysis of this system is straightforward, as long as we neglect any second-order effects of adjacent domains (note that these effects can in principle be significant). The presence of the 5T loop on either side of the non-orthogonal domains on the receptor is intended to reduce the significance of this. The following equations hold at equilibrium

$$K^t = \frac{[R_i{:}L_j]}{[R_i][L_j]} \tag{28}$$

$$K^a_{ij} = \frac{[(R_i{:}L_j)_a]}{[R_i{:}L_j]} = \frac{[(R_i{:}L_j)_{ab}]}{[(R_i{:}L_j)_b]} \tag{29}$$

$$K^b_{ij} = \frac{[(R_i{:}L_j)_b]}{(R_i{:}L_j]} = \frac{[(R_i{:}L_j)_{ab}]}{[(R_i{:}L_j)_a]} \tag{30}$$

and some algebra yields the following:

$$\sum_{1\le i\le n_R, 1\le j\le n_L} [(R_i{:}L_j)^*] = \sum_{1\le i\le n_R} \frac{[R_i]_0 \sum_{j=1}^{n_L} K^t(1+K^a_{ij})[L_j]}{1+\sum_{j=1}^{n_L} K^t(1+K^a_{ij})(1+K^b_{ij})[L_j]} \tag{31}$$

$$\approx \sum_{1\le i\le n_R} \frac{[R_i]_0 \sum_{j=1}^{n_L} K^t(1+K^a_{ij})[L_j]_0}{\sum_{j=1}^{n_L} K^t(1+K^a_{ij})(1+K^b_{ij})[L_j]_0} \tag{32}$$

where $[(R_i{:}L_j)^*] = [R_i{:}L_j] + [(R_i{:}L_j)_a]$ is the concentration of fluorescing receptor, noting that $[R_i] \approx 0$ because ligands are in excess so that essentially no receptor will be unbound.

Note the similarity to Equation (20). The two are formally equivalent if we make the following correspondences

$$K_{ij} = K^t(1+K^a_{ij})(1+K^b_{ij}) \tag{33}$$

$$\varepsilon_{ij} = \frac{1}{1+K^b_{ij}}. \tag{34}$$

Thus, this system is capable of the same signal responses as the ligand-receptor system of the previous section, and can in principle implement the four archetypal responses using two ligands and two receptors.

C Tables of sequences

Table 2 Base sequence and substitutions for the three designs D_x, D_y, and D_z.

cyclic graph	Base sequence	GCCTTGTATGTGAATATCCGTGTCA
	Fully mutated sequence	GCCATGAAAGAGTAAAACCGAGACA
Desargues graph	Base sequence	GGAGAATGATTAGCACGGAGAGTGG
	Fully mutated sequence	GGTGTAAGTTAAGCTCGGTGTGAGG
weighted graph	Base sequence	CGGTGTGCTTTTACTTAAGTAACCG
	Fully mutated sequence	CGGAGAGCATATTCATTAGAATCCG

Table 3 Sequences for the cyclic graph on 18 nodes (D_x), Desargues graph (D_y), and weighted graph (D_z).

Design	Name	Binary string	Sequence
cyclic	x_0	000000000	GCCTTGTATGTGAATATCCGTGTCA
	x_1	100000000	GCCATGTATGTGAATATCCGTGTCA
	x_2	110000000	GCCATGAATGTGAATATCCGTGTCA
	x_3	111000000	GCCATGAAAGTGAATATCCGTGTCA
	x_4	111100000	GCCATGAAAGAGAATATCCGTGTCA
	x_5	111110000	GCCATGAAAGAGTATATCCGTGTCA
	x_6	111111000	GCCATGAAAGAGTAAATCCGTGTCA
	x_7	111111100	GCCATGAAAGAGTAAAACCGTGTCA
	x_8	111111110	GCCATGAAAGAGTAAAACCGAGTCA
	x_9	111111111	GCCATGAAAGAGTAAAACCGAGACA
	x_{10}	011111111	GCCTTGAAAGAGTAAAACCGAGACA
	x_{11}	001111111	GCCTTGTAAGAGTAAAACCGAGACA
	x_{12}	000111111	GCCTTGTATGAGTAAAACCGAGACA
	x_{13}	000011111	GCCTTGTATGTGTAAAACCGAGACA
	x_{14}	000001111	GCCTTGTATGTGAAAAACCGAGACA
	x_{15}	000000111	GCCTTGTATGTGAATAACCGAGACA
	x_{16}	000000011	GCCTTGTATGTGAATATCCGAGACA
	x_{17}	000000001	GCCTTGTATGTGAATATCCGTGACA
Desargues	y_0	10100	GGTGAAAGATTAGCACGGAGAGTGG
	y_1	10110	GGTGAAAGTTTAGCACGGAGAGTGG
	y_2	10010	GGTGAATGTTTAGCACGGAGAGTGG
	y_3	11010	GGTGTATGTTTAGCACGGAGAGTGG
	y_4	01010	GGAGTATGTTTAGCACGGAGAGTGG
	y_5	01011	GGAGTATGTTAAGCACGGAGAGTGG
	y_6	01001	GGAGTATGATAAGCACGGAGAGTGG
	y_7	01101	GGAGTAAGATAAGCACGGAGAGTGG
	y_8	00101	GGAGAAAGATAAGCACGGAGAGTGG
	y_9	10101	GGTGAAAGATAAGCACGGAGAGTGG
	y_{10}	11100	GGTGTAAGATTAGCACGGAGAGTGG
	y_{11}	00110	GGAGAAAGTTTAGCACGGAGAGTGG
	y_{12}	10011	GGTGAATGTTAAGCACGGAGAGTGG
	y_{13}	11000	GGTGTATGATTAGCACGGAGAGTGG
	y_{14}	01110	GGAGTAAGTTTAGCACGGAGAGTGG
	y_{15}	00011	GGAGAATGTTAAGCACGGAGAGTGG
	y_{16}	11001	GGTGTATGATAAGCACGGAGAGTGG
	y_{17}	01100	GGAGTAAGATTAGCACGGAGAGTGG
	y_{18}	00111	GGAGAAAGTTAAGCACGGAGAGTGG
	y_{19}	10001	GGTGAATGATAAGCACGGAGAGTGG
weighted	z_0	000011110	CGGTGTGCTTTTTCATTAGAAACCG
	z_1	001100110	CGGTGTGCATATACTTTAGAAACCG
	z_2	010101010	CGGTGAGCTTATACATAAGAAACCG
	z_3	000000000	CGGTGTGCTTTTACTTAAGTAACCG
	z_4	011010010	CGGTGAGCATTTTCTTAAGAAACCG
	z_5	111111110	CGGAGAGCATATTCATTAGAAACCG
	z_6	000000001	CGGTGTGCTTTTACTTAAGTATCCG
	z_7	011010011	CGGTGAGCATTTTCTTAAGAATCCG
	z_8	111111111	CGGAGAGCATATTCATTAGAATCCG

Table 4 Shared sequences for the DNA strand displacement ligand-receptor networks.

Name	Binary string	Sequence
t		GAGAACT
n_{R_1}		GGTCTTGACAAACGTGTGCT
m_{R_1}		TATGAGGACGAATCTCCCGC
n_{R_2}		CCGATGTTGACGGACTAATC
m_{R_2}		GTTTATCGGGCGTGGTGCTC
a_{R_1}	111111111	GCCATGAAAGAGTAAAACCGAGACA
$a_{R'_1}$	111111100	GCCATGAAAGAGTAAAACCGTGTCA
a_{R_2}	111111100	GCCATGAAAGAGTAAAACCGTGTCA
$a_{R'_2}$	111111111	GCCATGAAAGAGTAAAACCGAGACA
b_{R_1}	000000000	GGAGAATGATTAGCACGGAGAGTGG
$b_{R'_1}$	100000000	GGTGAATGATTAGCACGGAGAGTGG
b_{R_2}	011111111	GGAGTAAGTTAAGCTCGGTGTGAGG
$b_{R'_2}$	111111111	GGTGTAAGTTAAGCTCGGTGTGAGG

Table 5 Gate-specific sequences for the DNA strand displacement ligand-receptor networks.

Gate	Name	Binary string	Sequence
Balance	a_{L_1}	111111100	GCCATGAAAGAGTAAAACCGTGTCA
	a_{L_2}	111111111	GCCATGAAAGAGTAAAACCGAGACA
	b_{L_1}	100000000	GGTGAATGATTAGCACGGAGAGTGG
	b_{L_2}	111111111	GGTGTAAGTTAAGCTCGGTGTGAGG
Imbalance	a_{L_1}	111111110	GCCATGAAAGAGTAAAACCGAGTCA
	a_{L_2}	111111110	GCCATGAAAGAGTAAAACCGAGTCA
	b_{L_1}	100000000	GGTGAATGATTAGCACGGAGAGTGG
	b_{L_2}	111111111	GGTGTAAGTTAAGCTCGGTGTGAGG
Ratiometric	a_{L_1}	111111110	GCCATGAAAGAGTAAAACCGAGTCA
	a_{L_2}	111111111	GCCATGAAAGAGTAAAACCGAGACA
	b_{L_1}	100000000	GGTGAATGATTAGCACGGAGAGTGG
	b_{L_2}	111111111	GGTGTAAGTTAAGCTCGGTGTGAGG
Additive	a_{L_1}	111111110	GCCATGAAAGAGTAAAACCGAGTCA
	a_{L_2}	111111110	GCCATGAAAGAGTAAAACCGAGTCA
	b_{L_1}	111111111	GGTGTAAGTTAAGCTCGGTGTGAGG
	b_{L_2}	111111111	GGTGTAAGTTAAGCTCGGTGTGAGG

Table 6 Sequences for the hairpin logic circuits. The full hairpin sequence is $al\bar{a}^*$ for signal A.

Gate	Name	Binary string	Sequence
	l		TTT
NOT	i	11111111	GGTGTAAGTTAAGCTCGGTGAGAGG
	$\bar{i}$	11110000	GGTGAAAGATTAGCACGGTGAGAGG
	a	11100000	GGTGAAAGATTAGCACGGAGAGAGG
	$\bar{a}$	11101111	GGTGTAAGTTAAGCTCGGAGAGAGG
	q	11001111	GGTGTAAGTTAAGCTCGGAGAGTGG
	$\bar{q}$	11000000	GGTGAAAGATTAGCACGGAGAGTGG
	s	00000000	GGAGAATGATTAGCACGGAGAGTGG
	$\bar{s}$	00001111	GGAGTATGTTAAGCTCGGAGAGTGG
AND	i_1	111000000	GGTGAAAGATTAGCACGGAGAGAGG
	$\bar{i}_1$	111000111	GGTGAAAGATAAGCTCGGAGTGAGG
	q_1	110000111	GGTGAAAGATAAGCTCGGAGTGTGG
	$\bar{q}_1$	110000000	GGTGAAAGATTAGCACGGAGAGTGG
	i_2	000111000	GGAGTATGTTTAGCACGGTGAGTGG
	$\bar{i}_2$	000111111	GGAGTATGTTAAGCTCGGTGTGTGG
	q_2	000110111	GGAGTATGATAAGCTCGGTGTGTGG
	$\bar{q}_2$	000110000	GGAGTATGATTAGCACGGTGAGTGG
	s	000000000	GGAGAATGATTAGCACGGAGAGTGG
	$\bar{s}$	000000111	GGAGAATGATAAGCTCGGAGTGTGG
Square Root	i_1	001010100	GGAGAAAGTTTAGCACGGAGAGAGG
	$\bar{i}_1$	110100100	GGAGTATGTTAAGCACGGAGTGTGG
	i_2	001110000	GGAGTAAGATTAGCACGGAGAGAGG
	$\bar{i}_2$	110000000	GGAGAATGATAAGCACGGAGTGTGG
	i_3	010001100	GGTGAATGTTAAGCACGGAGAGTGG
	$\bar{i}_3$	101111100	GGTGTAAGTTTAGCACGGAGTGAGG
	i_4	100001100	GGTGAATGTTTAGCACGGAGTGTGG
	$\bar{i}_4$	011111100	GGTGTAAGTTAAGCACGGAGAGAGG
	m_1	111100100	GGAGTATGTTAAGCACGGAGTGAGG
	$\bar{m}_1$	000010100	GGAGAAAGTTTAGCACGGAGAGTGG
	m_2	111000000	GGAGAATGATAAGCACGGAGTGAGG
	$\bar{m}_2$	000110000	GGAGTAAGATTAGCACGGAGAGTGG
	m_3	101111000	GGTGTAAGATTAGCACGGAGTGAGG
	$\bar{m}_3$	010001000	GGTGAATGATAAGCACGGAGAGTGG
	m_4	011111000	GGTGTAAGATAAGCACGGAGAGAGG
	$\bar{m}_4$	100001000	GGTGAATGATTAGCACGGAGTGTGG
	x_{12}	000010000	GGAGAAAGATTAGCACGGAGAGTGG
	$\bar{x}_{12}$	111100000	GGAGTATGATAAGCACGGAGTGAGG
	x_{23}	010000000	GGAGAATGATAAGCACGGAGAGTGG
	$\bar{x}_{23}$	101110000	GGAGTAAGATTAGCACGGAGTGAGG
	x_{24}	100000000	GGAGAATGATTAGCACGGAGTGTGG
	$\bar{x}_{24}$	011110000	GGAGTAAGATAAGCACGGAGAGAGG
	q_1	110000100	GGAGAATGTTAAGCACGGAGTGTGG
	$\bar{q}_1$	001110100	GGAGTAAGTTTAGCACGGAGAGAGG
	q_2	111110000	GGAGTAAGATAAGCACGGAGTGAGG
	$\bar{q}_2$	000000000	GGAGAATGATTAGCACGGAGAGTGG
	s_1	001100110	GGAGTATGTTTAGCACGGTGAGAGG
	$\bar{s}_1$	110010110	GGAGAAAGTTAAGCACGGTGTGTGG
	s_2	000000011	GGAGAATGATTAGCTCGGTGAGTGG
	$\bar{s}_2$	111110011	GGAGTAAGATAAGCTCGGTGTGAGG

Revisiting Hybridization Kinetics with Improved Elementary Step Simulation

Jordan Lovrod
University of British Columbia, Vancouver, Canada

Boyan Beronov
University of British Columbia, Vancouver, Canada

Chenwei Zhang
University of British Columbia, Vancouver, Canada

Erik Winfree
California Institute of Technology, Pasadena, CA, USA

Anne Condon
University of British Columbia, Vancouver, Canada

Abstract

Nucleic acid strands, which react by forming and breaking Watson-Crick base pairs, can be designed to form complex nanoscale structures or devices. Controlling such systems requires accurate predictions of the reaction rate and of the folding pathways of interacting strands. Simulators such as *Multistrand* model these kinetic properties using continuous-time Markov chains (CTMCs), whose states and transitions correspond to secondary structures and elementary base pair changes, respectively. The transient dynamics of a CTMC are determined by a kinetic model, which assigns transition rates to pairs of states, and the rate of a reaction can be estimated using the mean first passage time (MFPT) of its CTMC. However, use of *Multistrand* is limited by its slow runtime, particularly on rare events, and the quality of its rate predictions is compromised by a poorly-calibrated and simplistic kinetic model. The former limitation can be addressed by constructing truncated CTMCs, which only include a small subset of states and transitions, selected either manually or through simulation. As a first step to address the latter limitation, Bayesian posterior inference in an Arrhenius-type kinetic model was performed in earlier work, using a small experimental dataset of DNA reaction rates and a fixed set of manually truncated CTMCs, which we refer to as *Assumed Pathway* (AP) state spaces. In this work we extend this approach, by introducing a new prior model that is directly motivated by the physical meaning of the parameters and that is compatible with experimental measurements of elementary rates, and by using a larger dataset of 1105 reactions as well as larger truncated state spaces obtained from the recently introduced stochastic *Pathway Elaboration* (PE) method. We assess the quality of the resulting posterior distribution over kinetic parameters, as well as the quality of the posterior reaction rates predicted using AP and PE state spaces. Finally, we use the newly parameterised PE state spaces and *Multistrand* simulations to investigate the strong variation of helix hybridization reaction rates in a dataset of Hata et al. While we find strong evidence for the nucleation-zippering model of hybridization, in the classical sense that the rate-limiting phase is composed of elementary steps reaching a small "nucleus" of critical stability, the strongly sequence-dependent structure of the trajectory ensemble up to nucleation appears to be much richer than assumed in the model by Hata et al. In particular, rather than being dominated by the collision probability of nucleation sites, the trajectory segment between first binding and nucleation tends to visit numerous secondary structures involving misnucleation and hairpins, and has a sizeable effect on the probability of overcoming the nucleation barrier.

2012 ACM Subject Classification Applied computing → Chemistry

Keywords and phrases DNA reaction kinetics, kinetic model calibration, simulation-based Bayesian inference, continuous-time Markov chains

Digital Object Identifier 10.4230/LIPIcs.DNA.29.5

29th International Conference on DNA Computing and Molecular Programming (DNA 29).
Editors: Ho-Lin Chen and Constantine G. Evans; Article No. 5; pp. 5:1–5:24

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Nucleic acid strands, which can react *in vitro* or *in vivo* by forming and breaking Watson-Crick base pairs, can be designed to fold into specific two and three-dimensional nanostructures [19, 41] through methods such as tile and brick assembly [64, 13, 36] and DNA origami [54, 21]. These nanostructures can execute various chemical [65, 11], mechanical [45, 5, 6], computational [4, 63, 75], and biomedical tasks [62, 20, 12, 74, 80, 26, 40]. To design and debug such systems of interacting nucleic acid strands within their environment, it is valuable to predict both their thermodynamic behaviour (such as energetically stable structures) and their kinetic behaviour (such as folding pathways or reaction rates).

Efficient, general-purpose methods are available for predicting thermodynamic properties of interacting nucleic acid strands [34, 85, 76, 25]. These methods leverage "nearest-neighbor" models of nucleic acid thermodynamics that have been well calibrated from experimental data over many decades [46, 58, 57]. In contrast, general-purpose methods for predicting kinetic properties can be slow and inaccurate. One such simulation model, *Multistrand* [59, 60], samples folding trajectories from initial to final states through the space of all possible secondary structures of the nucleic acid system. Each step along the trajectory is an elementary transition [23], in which a base pair forms or breaks and in which a holding time is consumed. Such simulations can be prohibitively slow when the reaction is a rare event, e.g., when the initial and final states are separated by a high energy barrier. Moreover, *Multistrand*'s elementary rates are determined by a combination of nearest neighbor thermodynamics and a 2-parameter Metropolis kinetic model, which is too simplistic to produce reliable rate predictions.

Two important improvements address these limitations of *Multistrand*. First, the *Pathway Elaboration* (PE) method uses *Multistrand* to build truncated state spaces of reaction kinetics [83, 81]. These smaller state spaces are amenable to matrix methods for rate computation, which are efficient even on rare events. Second, a 15-parameter Arrhenius kinetic model takes into account the local context around each elementary step, which in principle should improve reaction rate predictions [82]. This Arrhenius model was calibrated using a dataset of a few hundred reactions, on small, customised state spaces, that reflect assumptions about likely pathways. We refer to these as *Assumed Pathway* (AP) state spaces.

1.1 Improved parameter inference

However, previous work stopped short of the challenging computational task of calibrating the Arrhenius model using the PE state spaces. In this work, we describe a Bayesian inference approach to this problem. Section 3.1 introduces a new prior distribution over the kinetic parameters, which is directly motivated by their physical meaning and which is compatible with experimental measurements of elementary rates. We use a dataset of 1105 reactions, as will be described in Section 4.2, that had previously been sourced, but only a small subset of which had been used for inference in past work. For each reaction in this dataset, we generated a truncated state space with AP, and also using the existing PE implementation when it completed within 7 days and with 10GB of guaranteed RAM on a single CPU core[1].

Despite a significantly higher computational cost, we find that the larger PE state spaces do not always lead to more accurate rate predictions than the small, manually designed AP state spaces. For posterior approximation, we apply the standard *random walk Metropolis*

[1] The majority of the runtime and memory footprint were caused by the unoptimised Python implementation of PE, rather than by the *Multistrand* stochastic simulations in its inner loops.

(RWM) algorithm. The resulting posterior approximations in Section 4.3, which are often multimodal, recover an expected correlation structure among the kinetic parameters. However, we also uncover severe numerical instability in the linear equation systems required for rate prediction. Due to numerous design limitations in the legacy software, a significant refactoring effort was required to implement the above extensions. In Section 4.4, we discuss several modifications to the kinetic model and inference methods that could improve inference quality. Full details of this work are described in Lovrod's MSc Thesis [43].

1.2 Case study

In Section 5, we use the mode of the new posterior to parameterise the *Multistrand* model, and present a case study based on the helix association data of Hata et al. [33]. Although the examined sequences have equal length and similar melting temperatures, the reported hybridization rates spread over more than two orders of magnitude. Our thermodynamic simulations confirm a correlation between the experimental rate and the expected number of free bases in the Boltzmann ensemble of unbound strands. However, our kinetic simulations suggest that the process of nucleation, in the non-probabilistic sense of reaching three consecutive correct inter-strand base pairs, is nontrivial and insufficiently explained by the time to first binding. We then employ *Multistrand*'s "first step mode" (FSM) to analyse the probabilistic and temporal behaviour of trajectories that start at the moment of initial binding and end either with dissociation or hybridization. In particular, we find a positive correlation between the proportion of successfully associating trajectories and the experimental rate. Moreover, the two-stranded complexes often spend extended periods of time exploring conformations with misnucleation and/or hairpins, and visualisations of the reactive FSM trajectory ensemble indicate a rich and strongly sequence-dependent structure, including a multimodal first passage time distribution for some reactions.

2 Background and related work

The DNA reactions we consider occur in systems with fixed experimental conditions (solution volume V, temperature T, and concentrations of Na^+ and Mg^{2+} ions). We often use the inverse temperature $\beta = \frac{1}{k_B T}$, where k_B is the Boltzmann constant. A nucleic acid reaction, in which DNA or RNA strands fold from one three-dimensional structure into another by forming and breaking base pairs, can be described at the secondary structure level with an initial microstate (or initial distribution over microstates) representing the reactants, and a final microstate (or final region of microstates) representing the products. The number of microstates may in general scale exponentially in the total strand length l.

A thermodynamic model defines the Gibbs free energy $\Delta G(x)$ relative to some reference state, at all allowed states $x \in \mathcal{X}$ of a system, and gives rise to the Gibbs-Boltzmann distribution π and its partition function Z_β at inverse temperature β,

$$\pi(x \mid \beta) = \frac{1}{Z_\beta} \cdot e^{-\beta \cdot \Delta G(x)}, \qquad Z_\beta = \int e^{-\beta \cdot \Delta G(x)} dx, \tag{1}$$

which can be used to compute all quantities of interest at thermodynamic equilibrium.

2.1 Kinetic models of nucleic acid elementary steps

We focus in this work on elementary step models of nucleic acid reactions [23, 15, 59, 60, 22, 82], which offer a relatively fine-grained view of the system, with states and transitions corresponding to secondary structures and isolated changes in base pairs, respectively. These

simulators model the reaction kinetics using a *continuous-time Markov chain* (CTMC) on this state space, which will be defined in the next section. Notably, *Multistrand* can model multi-stranded reactions, but like other currently available elementary step models, it only allows pseudoknot-free secondary structures.

2.1.1 CTMCs and their mean first passage times (MFPTs)

A finite CTMC is characterised by an initial probability distribution π_0 over a finite set of allowed states $\mathcal{X}$ and a *transition rate matrix* $K : \mathcal{X}^2 \to \mathbb{R}$, such that for all pairs of distinct states x and x', $K(x, x')$ is the instantaneous transition rate from x to x', and $K(x, x) = -\sum_{x' \in \mathcal{X} \setminus x} K(x, x')$ [69]. We refer to the subset of states $I \subset \mathcal{X}$ with non-zero initial probability π_0 as the *initial region*. The *transition probability matrix* $P : \mathcal{X}^2 \to [0, 1]$ is the normalised rate matrix, $P(x, x') = -\frac{K(x,x')}{K(x,x)}$, whereas the *transition matrices* (or *propagators*) $Q_t = e^{tK}$ for $t \in \mathbb{R}_{\geq 0}$ determine the transient dynamics of a CTMC. A stochastic process $\{X(t)\}_{t \in \mathbb{R}_{\geq 0}}$ can then be identified with this CTMC if and only if

$$\mathbb{P}\left(X(t_0) = x_0, \ldots, X(t_n) = x_n\right) = \pi_0(x_0) \prod_{m \in [0, n-1]} Q_{t_{m+1} - t_m}(x_m, x_{m+1}) \tag{2}$$

holds for any $n \in \mathbb{N}_0$, $t_0 < \cdots < t_n \in \mathbb{R}_{\geq 0}$, and $x_0, \ldots, x_n \in \mathcal{X}$. In other words, trajectory probabilities can be arbitrarily decomposed into segment probabilities due to Markovianity, and the probability of any segment is determined solely by its spatial endpoints (x_m, x_{m+1}), its temporal endpoints (t_m, t_{m+1}) and the transition rate matrix.

For a fixed final region $F \subset \mathcal{X}$, the *mean first passage time* (MFPT) $\tau_F : \mathcal{X} \to \mathbb{R}_{\geq 0}$ denotes the expected time to reach F for the first time from each state. It satisfies

$$-\tau_F(x) \cdot K(x, x) = 1 + \sum_{x' \in \mathcal{X} \setminus x} \tau_F(x') \cdot K(x, x') \quad \text{for all} \quad x \in \mathcal{X} \setminus \{F\}\,, \tag{3}$$

which is a numerically solvable matrix equation for sufficiently small CTMCs in which all states are connected to F [69], and this is the approach taken in this work. The MFPT from I to F is then defined by taking the expectation over the initial state distribution. When the state space of a CTMC is large, it can be infeasible to use matrix methods to exactly compute quantities such as the MFPT. One can instead resort to Monte Carlo estimation, e.g., via the *stochastic simulation algorithm* (SSA) [31], although SSA can be prohibitively inefficient for rare event simulation such as in systems with several metastable regions.

2.1.2 Functional form of CTMC rates

A kinetic model with free parameters θ describes the non-equilibrium dynamics of a CTMC via transition rates. We classify elementary transitions into three distinct types: *association*, in which a base pair is formed between two previously separate complexes, *dissociation*, in which a base pair breaks and causes a complex with multiple strands to separate into two distinct complexes, and *isomerisation*, in which a base pair is formed/broken within a complex. The kinetic model in *Multistrand* [59, 60] can be expressed as

$$\ln K(x, x'|\beta, \theta) = \ln \bar{K}(x, x'|\beta, \theta) + \begin{cases} -\beta \cdot \mathbb{1}_{\Delta G(x') \geq \Delta G(x)}(\Delta G(x') - \Delta G(x)), & \text{isomerisation} \\ \ln(u \cdot \alpha_\theta), & \text{association} \\ \ln(u \cdot \alpha_\theta) - \beta \cdot (\Delta G(x') - \Delta G(x)), & \text{dissociation} \end{cases} \tag{4}$$

where the base transition rate function $\bar{K} : \mathcal{X}^2 \to \mathbb{R}_{\geq 0}$ and the bimolecular scaling parameter α_θ are parametric choices, x and x' are adjacent microstates, and u is the initial strand

concentration. A functional form of $\bar{K}(x, x'|\beta, \theta)$ which is symmetric with respect to x and x' can easily be shown to satisfy *detailed balance*,

$$\frac{K(x, x' \mid \beta, \theta)}{K(x', x \mid \beta, \theta)} = \frac{\pi(x' \mid \beta)}{\pi(x \mid \beta)} = e^{-\beta \cdot (\Delta G(x') - \Delta G(x))} \quad \text{for all} \quad x, x' \in \mathcal{X}\,, \tag{5}$$

which is a sufficient condition for recovering the Gibbs-Boltzmann distribution (1) in the steady-state limit.

2.1.3 Context-dependent Arrhenius rates

Kinetic models of Arrhenius type factorise the rate coefficient using a *pre-exponential factor* A and an *activation energy* E that couples to the temperature. They can be parameterised in ways that make the kinetic behavior of an elementary step dependent on local context features surrounding the affected base pair [82, 25], e.g., by assuming multiplicative (log-additive) effects selected by $C : \mathcal{X}^2 \to 2^{\mathcal{C}}$ from a set of transition context features $\mathcal{C}$,

$$\ln \bar{K}(x, x'|\theta, \beta) := \sum_{c \in C(x,x')} \ln A_{\theta,c} - \beta \sum_{c \in C(x,x')} E_{\theta,c}\,, \quad \theta := (\ln \alpha_\theta, (\ln A_{\theta,c})_{c\in\mathcal{C}}, (E_{\theta,c})_{c\in\mathcal{C}})\,. \tag{6}$$

In the Arrhenius model we consider in this work, $\mathcal{C}$ comprises topological *half contexts*, which refer to base pairing structures on a single side of the affected base pair in an elementary transition [82]. Symmetry of $\ln \bar{K}$ is ensured by applying the same features to the forward and reverse directions of a transition. This model differentiates between seven half contexts[2] which categorise transitions into 28 equivalence classes of *local contexts*[3]. It therefore contains 15 free kinetic parameters, whereas the Metropolis kinetic model originally used in *Multistrand* [59, 60] has only two, $(k_{\text{uni}}, k_{\text{bi}}) \equiv (A_\theta, \alpha_\theta \cdot A_\theta)$.

3 Bayesian model for kinetic parameters

3.1 Prior over kinetic parameters

Our new prior imposes an independent, weak log-normal distribution on the multiplicative rate contribution from each kinetic parameter dimension,

$$\begin{aligned} \ln \alpha_\theta &\sim \mathcal{N}\left(\mu = -2.3,\ \sigma^2 = 800\right)\left[\ln\left(M^{-1}\right)\right], \\ E_{\theta,c} &\sim \mathcal{N}\left(\mu = 0.0,\ \sigma^2 = 25\right)\left[\text{kcal/mol}\right], \quad \ln A_{\theta,c} \sim \mathcal{N}\left(\mu = 7.5,\ \sigma^2 = 110\right)\left[\ln s^{-1/2}\right]. \end{aligned} \tag{7}$$

Assuming fixed thermodynamic parameters, this prior effectively describes a temperature-dependent Gaussian law over the elementary log-rates, and provides support for all values that may be physically possible. It leads to $\ln \bar{K} \sim \mathcal{N}(15.0, 364.8)$ at 25°C and $\ln \bar{K} \sim \mathcal{N}(15.0, 342.3)$ at 50°C, which is compatible with experimental measurements of elementary rates [47, 15] as well as values from past calibration of the Metropolis model [59, 60, 68, 82, 83].

Notably, the pre-exponential factors A_c have non-negative support, and the particularly weak prior for $\ln \alpha$ is centred at $-\ln(10)$, which corresponds to the assumption that the numerical value of the bimolecular elementary rate coefficient is approximately one order of magnitude smaller than the numerical value of the unimolecular rate coefficient, when measured in the standard units of $M^{-1}s^{-1}$ and s^{-1}, respectively [59, 60, 68, 82, 83]. Our weak

[2] $\mathcal{C} :=$ {stack, loop, end, stack+loop, stack+end, loop+end, stack+stack}.
[3] The number of unordered pairs of half contexts is the multiset coefficient $\left(\!\binom{7}{2}\!\right) = \binom{8}{2} = 28$.

assumptions for the parameters E_c are that each elementary scale Arrhenius activation energy should be less, in magnitude, than experimentally estimated macroscale activation energies [9, 3, 50, 52], and that the elementary scale and macroscale activation energies will be closer in value for simple reactions such as hairpin closing/opening, helix association/dissociation and bubble closing, especially when the strands are short. Furthermore, centering E_c at zero amounts to a regularisation towards the functional form of the original Metropolis model.

3.2 Likelihood of observations

Our observation model for macroscopic rates follows an approach that was previously used for posterior inference [82] and for maximum likelihood estimation [82, 83] of elementary step rates. Given a kinetic parameterisation θ and a state space $\mathcal{X}$, a *deterministic* rate prediction for a reaction is simulated as the inverse of the expected MFPT τ_F from the initial state distribution π_0, representing reactants, to the non-empty final region F, representing products. More precisely, the rate coefficient prediction $\hat{k}$ depends on the MFPT as:

$$\hat{k} = \begin{cases} \dfrac{1}{\mathbb{E}_{\pi_0}[\tau_F \mid \mathcal{X}, \beta, \theta]} & \text{for reactions of the form } A \to B \text{ or } A \to B + C \\ \dfrac{1}{u\ \mathbb{E}_{\pi_0}[\tau_F \mid \mathcal{X}, \beta, \theta]} & \text{for reactions of the form } A + B \to C \text{ or } A + B \to C + D. \end{cases} \tag{8}$$

The former case describes unimolecular reactions, for which $\hat{k}$ has the meaning of a first-order reaction rate coefficient. Its estimate is based on the assumption that there are no intermediate association steps along the reaction pathway, which holds in our dataset. The latter case describes bimolecular reactions, for which $\hat{k}$ has the meaning of a second-order reaction rate coefficient[4]. By dimensional analysis, it requires a concentration quantity to relate to a time quantity. In general, second-order rates cannot be determined directly from the CTMC model of the *Multistrand* simulator, but the simple estimate above, which uses the initial concentration u of reactants, is justified in the limit of low concentrations, where the initial association step of second order is rate-limiting for the full reaction[5].

For each reaction r, our noise model then centers a log-normal distribution at the predicted rate coefficient to obtain a probabilistic synthetic observation for the log-rate coefficient, i.e., $\log_{10} k_r \sim \mathcal{N}(\log_{10} \hat{k}_r, \sigma_r^2)$, where the variance σ_r^2 is taken as the experimental variance of the log-rate coefficients among the reactions of the same *group*. A group of reactions is defined by its experimental publication and reaction type, and corresponds to a row in Appendix Table 3. In the case where $\hat{k}$ is a non-physical prediction, which occurs when the sparse solver for the MFPT fails or produces a negative or infinite prediction, we instead apply a constant likelihood close to zero, $\mathcal{N}(5\sigma_r^2 \mid 0, \sigma_r^2)$. It should be noted that this noise model is an *ad hoc* choice for constructing a likelihood kernel and, in its current form, cannot be understood as a physically motivated generative model.

[4] Note that the experimental rate coefficients in our dataset were not necessarily computed under the same assumptions, and, for instance, we have several strand displacement reactions in our dataset whose rate coefficients were estimated with first order fits. This warrants reconsideration in future work, particularly if new reactions at higher concentrations are included in the dataset.

[5] See [59, Ch. 7,8] for a discussion about estimating second-order rates from *Multistrand* simulation statistics, and [82, Sec. 5.2] for the modelling assumptions in (8).

3.3 Approximations of the intractable likelihood

In order to *truncate* the intractable state space underlying the *Multistrand* simulator, we employ the deterministic method in [82], which we call *Assumed Pathway* (AP), and the more recent stochastic method *Pathway Elaboration* (PE) [83]. A distinct truncated state space $\mathcal{X}$ is precomputed once for each reaction and is treated as fixed during inference.

The states included in the AP approximation are the non-pseudoknotted secondary structures whose base pairs occur in either the initial or final secondary structure, and which are reachable by elementary step transitions where the affected base pair is always at the boundary of a hybridized domain. Thus the AP method only considers a small subset of states and transitions which are assumed to cover the dominant pathway. For instance, in a helix association reaction, any hairpin formations prior to or during helix association would not be modelled, although they could significantly affect reaction rates [27]. To avoid these sorts of limitations, the PE method constructs truncated CTMCs stochastically, using the states found through a succession of distance-biased and unbiased trajectory samples [83].

The criteria for the initial and final regions in our simulations are reported in Appendix Table 1. In many cases, these criteria define regions that are much broader than those considered in previous work [83]. The initial states are treated as Boltzmann distributed according to the thermodynamic model, for which we use Nupack 3.2.2 [76]. The number of states in the PE approximations that satisfy the criteria for our endpoint regions is stochastic: Our simulations led to initial regions with up to 3689 states and final regions with up to 1080 states. In contrast, the AP state spaces include exactly one initial and one final state.

The computational cost of the PE method depends strongly on the choice of hyperparameters and on the size and energy landscape of the true state space. We aim to construct truncated CTMCs using a set of hyperparameters suggested in the original reference ($n_b = 128$, $n_\kappa = 256$, $b = 0.4$, $\kappa = 16ns$) [83]. Each CTMC construction attempt for all reactions in our dataset is given 7 days with at least 10GB of RAM on a single CPU core. However, these resources proved insufficient for our initial choice of hyperparameters for many reactions. We therefore attempted eight different hyperparameter settings, which we report and name in Appendix Table 2. We give preference to the hyperparameter settings in order of the expected resulting state space size. We did not use the δ-pruning step of the PE method, because, in the existing implementation, its computational cost incurred during the CTMC construction did not warrant the speedup that could have been achieved in inference. All PE state spaces were generated with the Metropolis kinetic parameters

$$k_{uni} = 3.61 \times 10^6 \ [s^{-1}], \quad k_{bi} = 1.12 \times 10^5 \ [M^{-1}\, s^{-1}], \tag{9}$$

which are equivalent to the Arrhenius parameters

$$\ln \alpha = -3.47 \ [\ln M^{-1}], \quad \forall c \in \mathcal{C}.\ \ln A_c = 7.55 \ [\ln s^{-1/2}], \quad E_c = 0 \ [\text{kcal/mol}] \tag{10}$$

in the sense of Section 2.1.3. These parameters were the result of previous parameter tuning on PE state spaces via gradient-free maximum a posteriori (MAP) optimisation [83].

4 Bayesian inference for kinetic parameters

4.1 Bayesian inference methods

Following the approach taken in [82], we used Markov chain Monte Carlo (MCMC) [18, 53, 42] for approximate Bayesian inference, but with a different implementation choice. In particular, our Bayesian model was expressed in the probabilistic programming framework PyMC [56], using custom operations to construct and solve the sparse linear equations for the MFPT,

and the *random walk Metropolis* (RWM) algorithm [48] was used for inference. In addition to the inference algorithm implementation, PyMC provides a number of standard tools [7, 55] for diagnosing the behaviour of MCMC samplers, e.g., trace plots and the *effective sample size* (ESS) estimated using Geyer's initial monotone sequence criterion [29, 30].

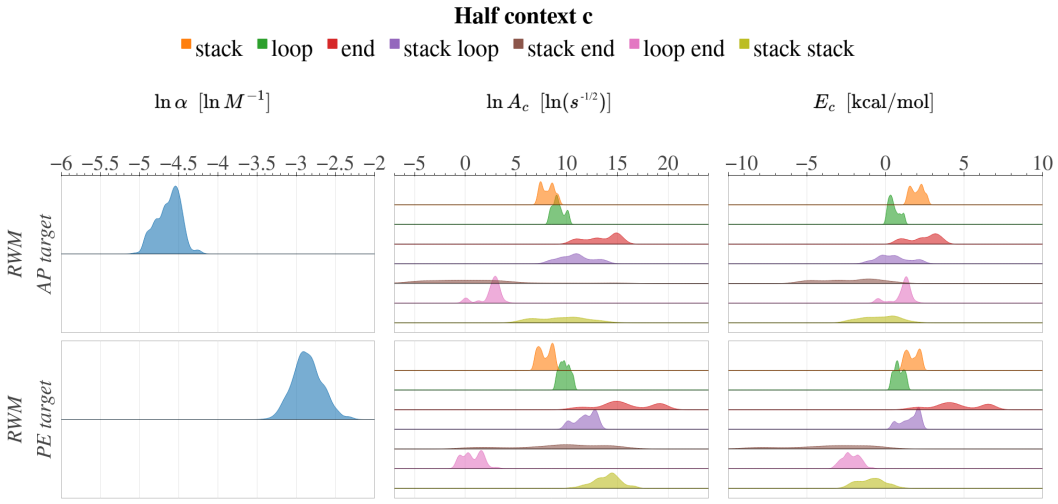


Figure 1 Posterior densities over the Arrhenius kinetic parameters in Section 2.1.3. Approximations are obtained from 600 samples using RWM, on AP and PE inference targets as defined in Section 4.2. The corresponding trace plots are shown in Appendix Figure 7.

4.2 Dataset and likelihood approximation results

The dataset used for parameter inference, summarised in Appendix Table 3, consists of 1105 DNA reactions and includes hairpin opening/closing [9, 8, 37], bubble closing [3], helix association/dissociation [49, 52], and three-way strand displacement [52, 77, 44][6]. We define the following two posterior inference targets, which use different combinations of data subsets and state space approximations.

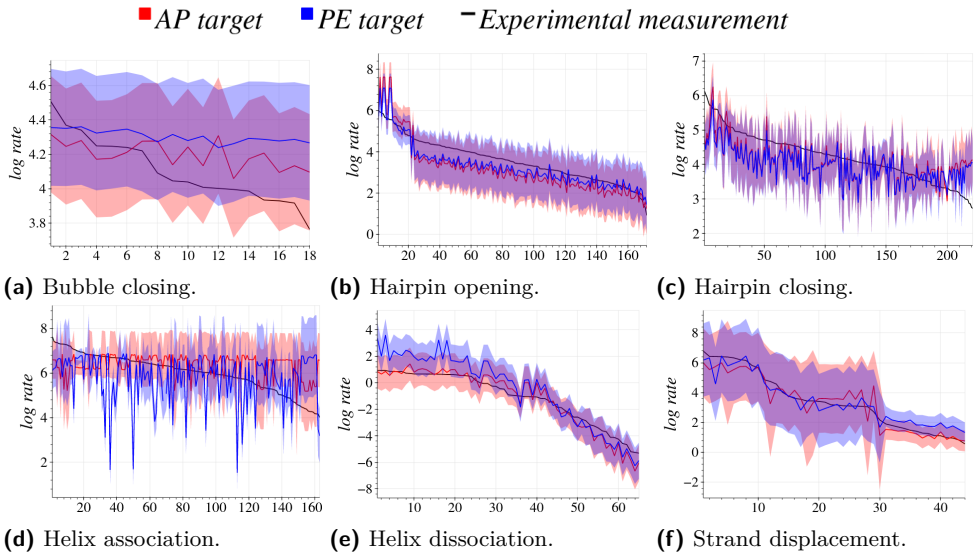
1. *AP target*: All 1105 reactions, using state spaces constructed by the AP method.
2. *PE target*: Only 683 reactions, using all the valid CTMCs that could be constructed by the existing PE implementation, with a preference over hyperparameter values as described in Section 3.3.

These inference targets allow us to indirectly compare the state space approximation methods through their behaviour during inference. For each inference target, the dataset and state spaces are summarised in Appendix Table 4. The *stack* and *loop* half contexts together account for more than 80% of the half context occurrences in each group of truncated CTMCs.

4.3 Posterior approximation results

We ran RWM for 800 total steps, 200 of which were discarded as burn-in. These choices, which proved sufficient for an analysis of the current model and its most significant bottlenecks, were motivated by computational resource constraints and past posterior inference attempts [83].

[6] The dataset of reaction rate coefficients was collected by Sedigheh Zolaktaf. A small set of 14 four-way strand exchange reactions were also collected [15], which we exclude due to computational limitations of the PE implementation. Small subsets of the collected data have been used for parameter tuning or Bayesian inference in previous work [82, 84, 83].



(a) Bubble closing. **(b)** Hairpin opening. **(c)** Hairpin closing.

(d) Helix association. **(e)** Helix dissociation. **(f)** Strand displacement.

Figure 2 Posterior predictive distributions for each reaction type, using the AP and PE likelihood approximations and the corresponding posteriors in Figure 1. The horizontal axis is the reaction index, ordered by decreasing experimental reaction rate. Red and blue solid lines show the mean of the posterior predicted log-rates, and the shaded regions are the 4-96 percentile ranges. We only display the 683 reactions that appear in both inference targets. Both predictive distributions were approximated by taking 100 samples from the likelihood kernel for each posterior sample.

The hyperparameter settings for the RWM sampler were chosen such that approximately the same number of sparse matrix solves are performed in each posterior inference attempt. We designated the kinetic parameters in Equation (10), which are in the 2-dimensional subspace of the Metropolis model, as the seed for all MCMC experiments, expecting that this initial point will mostly yield CTMCs that are physically possible and numerically stable.

The resulting posterior densities are shown in Figure 1. While distinctly multimodal in many dimensions, the shapes of the marginal posterior approximations for each $\ln A_c$ roughly mirror the shapes of the corresponding E_c, indicating a strong correlation between the kinetic parameters associated with the same half context. When comparing posterior results from different chains, the bimolecular scaling parameter $\ln \alpha$ appears to be multimodal, and seems to correlate with the *loop-end* and *stack-stack* half contexts, which are of low frequency in our CTMCs. There has only been one other reported attempt at Bayesian inference on the Arrhenius parameters [82], which used a smaller dataset of 376 reactions and AP state spaces. Despite our different prior, likelihood width, dataset, and inference method, most of our high density intervals for the $\ln A_c$ and E_c dimensions overlap with those reported in [82]. Furthermore, their posterior correlation matrix reflected a correlation structure between corresponding $\ln A_c$ and E_c dimensions that is qualitatively similar to ours.

In general, the approximation quality of Bayesian inference is determined by a complex interaction of the inference method and its hyperparameters with the forward model and the dataset, and each component should be assessed as a potential cause for poor behaviour in inference [28]. As a first step of sampling diagnostics, the RWM trace plots in Appendix Figure 7 display varied behaviour across different half contexts. For example, the *stack-stack* parameters are consistently explored much more broadly than the *stack* and *loop* parameters. In contrast, predictive checks are useful for understanding a prior or a posterior in terms of the distribution of predictions it generates [10]. In Figure 2, we compare the posterior

predictive distributions from RWM using the AP and PE targets. Even though the PE state spaces often cover more of the energy landscape, their posterior predictive log-rates are not consistently more accurate than those resulting from the AP state spaces. For further quantitative analyses of the inference results we refer to [43].

4.4 Discussion

4.4.1 Next steps for the kinetic model

The kinetic parameter dimensions that are least explored in our RWM chains (Appendix Figure 7, rows 1-3) roughly correspond to the half contexts that occur most frequently. This might suggest that some of the half contexts are underspecified, and that the multimodalities in the posterior approximations (Figure 1, rows 1-3) arise from significant differences in kinetic behaviour based on features that are not made explicit by the current kinetic model. It would therefore be worth considering kinetic models that partition the transitions into more fine-grained equivalence classes. For instance, the half contexts could be defined in a way that accounts for stack and loop sizes, or for base identities. These refinements could also improve posterior rate predictions, particularly for hairpin closing reactions, whose rate coefficients are not well-captured in our current model (see Figure 2c), and whose experimental rate measurements suggest sequence-dependent behavior [32].

4.4.2 Next steps for MCMC inference

Because RWM consistently recovers an expected correlation structure that is not incorporated into the prior or the proposal, significant sampling effort is spent discovering these correlations. The sample efficiency could be improved by using *Gibbs sampling*, in which a single sample from the target is constructed by iteratively drawing from the conditional distributions of parameter dimensions, while treating all other parameters as observed. Sampling from such intractable conditional distributions is often performed via nested Metropolis-Hastings steps, and the procedure is known as *Metropolis within Gibbs*. It might also be beneficial to partition the parameters into *Gibbs blocks*, containing subsets of mutually highly correlated parameters which are proposed jointly. In our case, the RWM posteriors suggest grouping $(\ln A_{\theta,c}, E_{\theta,c})$ for each $c \in \mathcal{C}$.

4.4.3 Next steps for the likelihood formulation

Within the high density intervals of our posterior approximations, the posterior rate predictions are more strongly influenced by the state space approximation than by the kinetic parameters. This suggests that our current likelihood model cannot further distinguish between different kinetic parameters on the state spaces considered. Moreover, while the PE state spaces cover a higher proportion of the full energy landscapes than the AP state spaces, they do not consistently yield more accurate posterior rate predictions, as indicated by Figure 2. We attribute this finding primarily to the high proportion of sparse linear solver failures or non-physical solutions during inference, which arise from ill-conditioned MFPT equations and for which we apply a constant likelihood close to zero (see Section 3.2). Hence, while the MFPT equations evaluated at our posterior samples yield valid solutions, we expect that the truncation-dependent and solver-dependent likelihood term biases the information extracted during inference from the experimental data. A more detailed quantification of the numerical issues around PE, MFPT equations and posterior inference can be found in [43].

This bottleneck cannot easily be resolved by expanding the dataset or by increasing the approximation quality of the truncated state spaces. The numerical stability of the MFPT computation could in principle be improved by using an iterative solver with suitable preconditioning techniques that might render more of the systems solvable. However, regardless of the preconditioner, the current way of estimating the rate coefficients via the MFPT is a simple, scalar observation model which conceals some transient kinetic effects and which ignores the variety of regression techniques used to estimate reaction rates from experimental observations. Hence, in addition to improved kinetic model features and state space approximations, a reformulation of the observation model to better incorporate transient observations appears prudent.

4.4.4 Software

Multistrand is an efficient and general forward simulator, but its implementation is not amenable to the inverse problem of parameter inference and does not support flexible parallelism. Any task that would require online updates to the state probabilities or transition rates cannot be achieved without considerable overhead in software development and resource usage. This includes forward simulation via replica exchange, Bayesian model averaging, and kinetic parameter inference methods that sample trajectories in an inner loop.

Python libraries used for the work in Section 4 include PyMC [56], Aesara [17, 70], ArviZ [38], Xarray [35], Joblib [71], SciPy, and UMFPACK [72, 16]. At the time of publication of this manuscript, a new official minor release of *Multistrand* will port it to Python 3 and will provide an Apptainer/Singularity container [39], making it simple to run on Linux host systems including HPC clusters. This will be accompanied by a software release specific to this work[7], consisting of the posterior approximations in Section 4.3 and the post-processing scripts used for the *NUPACK* and *Multistrand* simulations in Section 5.

5 Case study

5.1 Motivation

Among the reaction types in the dataset, the most pronounced difference between the predictions made by the AP and PE state spaces is on the helix association reactions in Figure 2d, suggesting that the additional secondary structures in the latter significantly affect kinetic behavior. Many studies have assessed secondary structure effects on the reaction rate of hybridization [27, 61, 33], although they typically extrapolate kinetic behaviour from thermodynamic properties.

This case study focuses on a recent set of 47 hybridization reactions by Hata et al. [33], which we will assess using PE state spaces and the mode of our new posterior distribution over kinetic parameters. The examined sequences have equal length and similar melting temperatures, and were designed to avoid very stable secondary structures as well as stable misnucleation and mishybridization. Nevertheless, at a fixed temperature of 25° C and single-strand concentration of 50 nM, the empirically estimated rate constants varied by more than two orders of magnitude. The authors suggest that decreases in hybridization rates can be explained by decreases in nucleation rates, caused by intra-molecular base pairs that, although not necessarily thermodynamically stable, render nucleation sites inaccessible.

[7] To be found at: `https://github.com/UBC-Mol-Prog/hybridization-profiling`

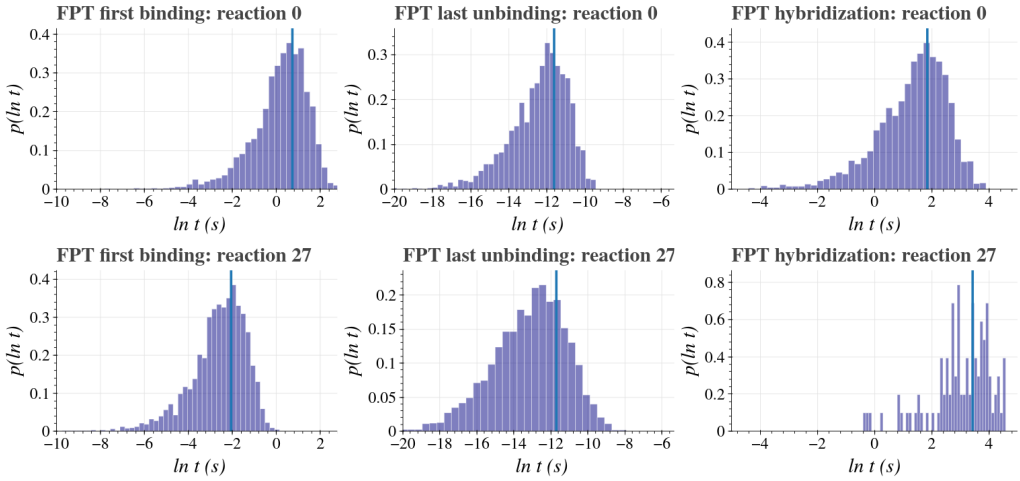


Figure 3 Samples of the first passage times for the first binding, last unbinding, and hybridization times for reactions no. 0 and no. 27 from [33]. The x-axis is the simulated ln-time in seconds, and the blue vertical lines indicate the simulated ln-MFPT. For each event, we attempted to gather 3000 samples using KPS within 24 hours, with 30GB of RAM and 20 CPUs. However, it was only possible to generate 102 samples for the hybridization of reaction no. 27.

The authors also estimated that mishybridized secondary structures (wherein an unwanted stack of successive base pairs forms between strands) were unstable and infrequent, and therefore that their effect on the overall kinetics was negligible.

To assess these hypotheses, we first employed the kinetic path sampling (KPS) implementation in DISCOTRESS [66, 67] to sample times from the first passage distributions for forming the first inter-strand bond, for breaking the only inter-strand bond, and for overall hybridization in the PE CTMCs. KPS is an enhanced sampling technique for rare event simulation, and requires a partition of states into communities, which we specified manually[8]. Results of the simulations on reactions no. 0 and no. 27 are given in Figure 3. Although the simulated first binding rate is much faster in reaction no. 27, its simulated total hybridization rate is much slower, which is compatible with experimental estimates. Furthermore, the rate of dissociating from a state with a single base pair is comparable between the two reactions. Therefore, although these rates are important in general, they do not consistently account for experimental differences in hybridization rates. We therefore expect significant sequence-dependent kinetic behavior to occur between the moments of first binding and stable nucleation.

5.2 Boltzmann statistics of initial states

Starting from the classical nucleation-zippering model [1, Ch. 8.2], Hata et al. emphasise the importance of single-strand secondary structures with positive Gibbs free energy of formation. In particular, they argue that the nucleation phase is rate-limiting, and propose an empirical model in which the hybridization rate is proportional to the expected effective nucleation site density of each strand. As a consistency check for this model and similar metrics over the Boltzmann distribution of single-stranded structures, we show in Figure 4 the average number of free bases per strand, the product over the average relative number of

[8] Secondary structures were binned according to the nearest (edit-distance) structure in the AP model.

nucleation sites in each strand (similar to Hata et al. [33]), and the average number of free and compatible (mis-)nucleation sites per strand pair. All three metrics demonstrate some positive correlation with the experimental rate, although the correlations appear too weak to validate a kinetic model for hybridization based solely on Boltzmann statistics over the unbound states.

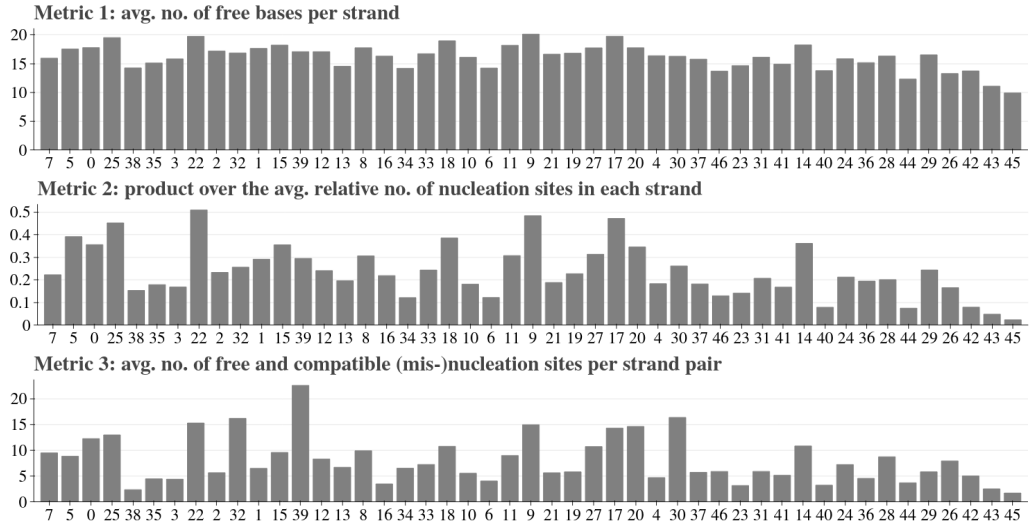


Figure 4 Secondary structure metrics[9] over the Boltzmann distribution of single-stranded structures, estimated from 10k samples of each strand in *NUPACK*. Reactions are ordered by decreasing experimental reaction rate and labeled using the same indices as in the reference [33].

5.3 Trajectory statistics after initial binding

Therefore, to assess potential secondary structure effects beyond the first moment at which the two complementary strands bind, we employ *Multistrand*'s "first step mode" (FSM), stopping the trajectory when the strands either fully dissociate or fully hybridize. Simulations are performed at the same strand concentration and temperature as the physical experiments. To analyze the trajectories, we define 8 different state types, which partition the secondary structures based on occurrences of hairpins[10], correctly hybridized stacks[11], and/or mishybridized stacks[12]. We label each of these types by three characters, which indicate whether there is at least one correctly hybridized stack (**S**) or not (**0**), at least one mishybridized stack (**M**) or not (**0**), and at least one hairpin (**H**) or not (**0**). For instance, **SM0** represents the type of states with at least three consecutive correctly hybridized base pairs, at least three consecutive mishybridized base pairs, and at most two consecutive intra-strand base pairs. Using these definitions, results of our kinetic simulations are provided in Figure 5. The proportion of first step trajectories ending in hybridization varies over

[9] **1.** $\frac{1}{2}\left(\mathbb{E}_{\pi_0^A}\left[n_b^A\right] + \mathbb{E}_{\pi_0^B}\left[n_b^B\right]\right)$, **2.** $\mathbb{E}_{\pi_0^A}\left[\frac{n_s^A}{n_s^{A,\max}}\right] \cdot \mathbb{E}_{\pi_0^B}\left[\frac{n_s^B}{n_s^{B,\max}}\right]$, **3.** $\mathbb{E}_{\pi_0^A \times \pi_0^B}\left[n_p^{A,B}\right]$, where for a strand A with complement B, π_0^A is the Boltzmann distribution over secondary structures, n_b^A is the no. of free bases, n_s^A is the no. of free nucleation sites, $n_s^{A,\max}$ is the max of n_s^A (and analogously for B), and $n_p^{A,B}$ is the no. of free and compatible (mis-)nucleation sites for the pair (A, B).

[10] 3+ consecutive base pairs occurring within a strand

[11] 3+ consecutive base pairs occurring between the two strands at the desired site

[12] 3+ consecutive base pairs occurring between the two strands in an undesired site

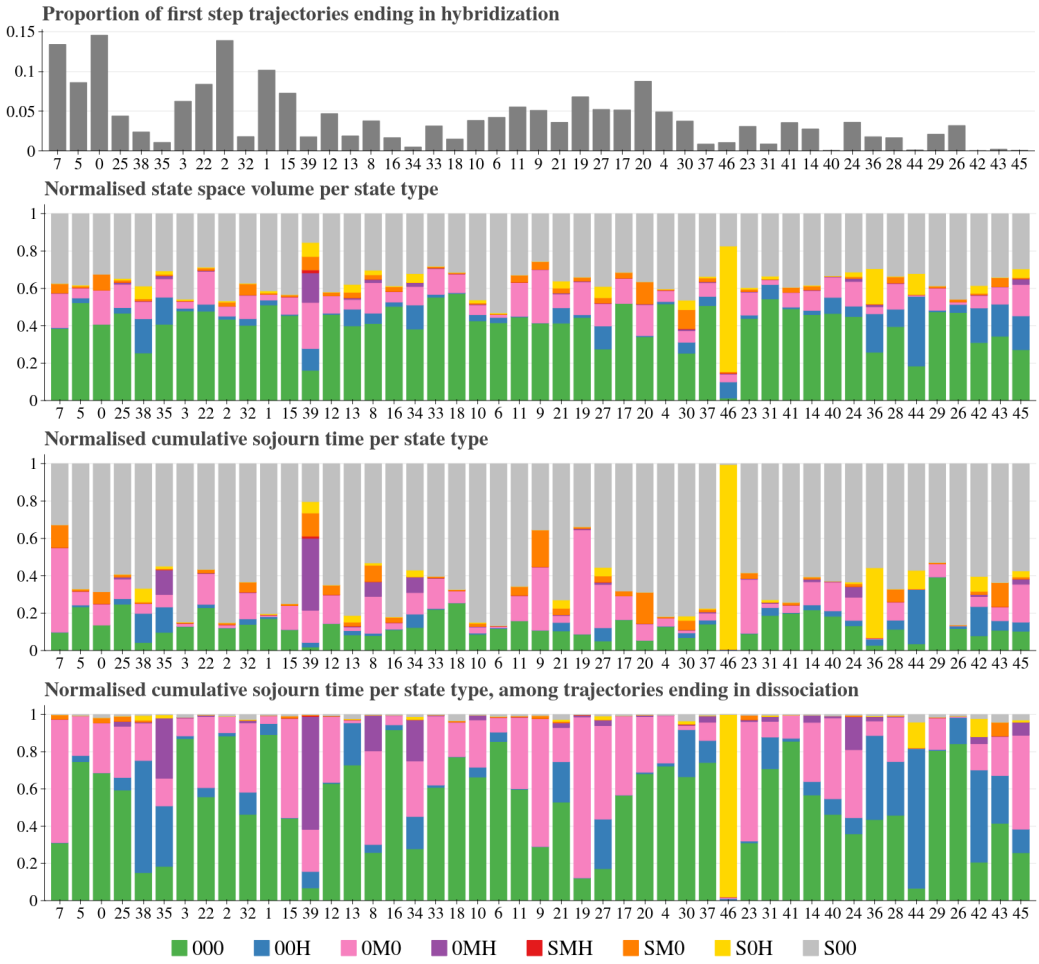


Figure 5 Results of FSM simulations in *Multistrand*. Reactions are ordered by decreasing experimental reaction rate. For each reaction, we accumulate reactive and non-reactive trajectory samples until at least 5 million elementary steps (maximum 155 million) and at least 500 hybridization trajectories (maximum 2305) have been sampled[13].

two orders of magnitude, although high hybridization proportions do not always correspond to fast reaction rates, such as in reaction no. 35. The proportion of different state types appears relatively consistent across different reactions, with some notable exceptions, such as reactions no. 39 and no. 46. In most reactions, we observe a high proportion of states with mishybridized stacks, which is contrary to one of the hypotheses in the experimental source [33]. Furthermore, the total time spent in conformations with misnucleation and/or hairpins is often significant, indicating that the mishybridized states, although potentially thermodynamically unfavourable, behave as kinetic traps. A small number of reactions, such as no. 1 and no. 3, appear to have folding pathways dominated by desired stacks, which is the underlying assumption in the AP model, while other reactions, such as no. 42 and no. 44, appear to have a high proportion of dissociating trajectories caused by simple hairpin formation after first binding, a phenomenon explored in other studies [27, 61].

[13] With the exception of reaction no. 46, whose hybridization trajectories are much longer. Our estimates therefore only include the 26 successful trajectories that could be stored using 30G of RAM, and contained 4.5 million secondary structures on average.

In Figure 6, the time spent in each state type is shown for 20 successful hybridization trajectories in six different reactions. These trajectories illustrate a high degree of sequence dependence, as well as a high variance of reactive pathways within reactions. Overall, these findings suggest that short sequence-level features, such as the nucleation capability proposed by Hata et al. [33], can be influential both before and after the first binding, and that in order to reach higher accuracy, simplified kinetic models of hybridization should in general consider misnucleation and hairpins in their choice of transition states. More generally, while the state predicates above are useful for assessing the reaction pathways from stochastic simulation, the correlation between the considered state predicates and the experimental rates appears too weak, and the relative strength of the considered kinetic effects too varied across reactions, to motivate or validate simplified mechanistic models for hybridization that are derived solely from state-based features. This reinforces the need for elementary step kinetic simulation methods that directly enable the parameterisation and estimation of local and global transient behaviour.

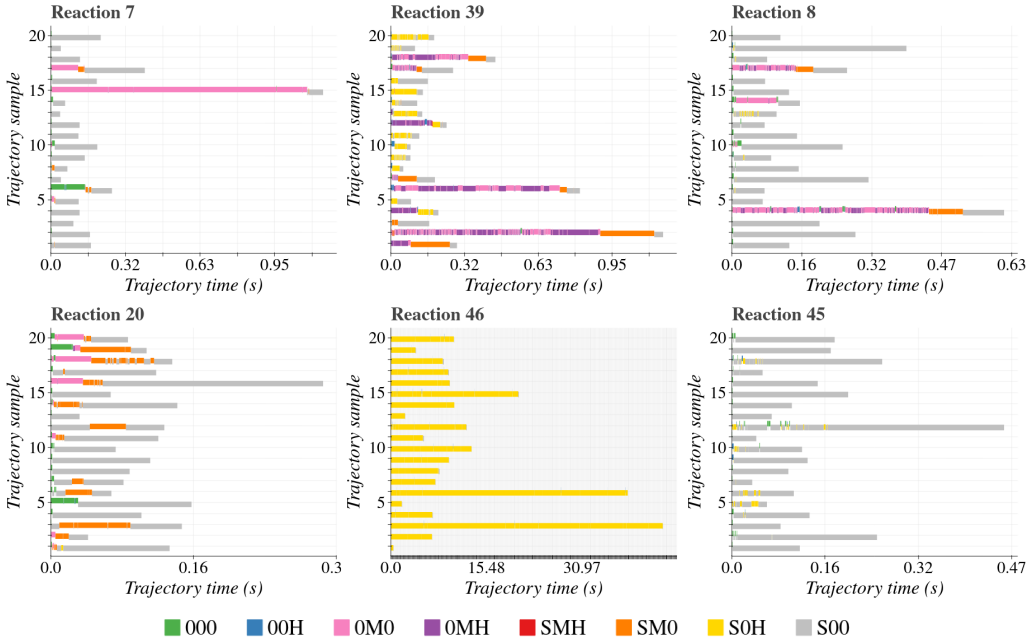


Figure 6 Examples of successful hybridization trajectories sampled using *Multistrand*'s FSM.

6 Conclusion and outlook

In this work, we extended a previous Bayesian inference approach to the calibration problem of an Arrhenius-type model of elementary step DNA kinetics. We introduced a new prior distribution over the desired kinetic parameters and expanded the training dataset to over 1000 reactions, using truncated state spaces generated with the *Assumed Pathway* (AP) method for the full dataset, and the *Pathway Elaboration* (PE) approach when computationally feasible. Posterior inference was performed with the *random walk Metropolis* (RWM) algorithm.

Our posterior distributions, though only preliminary in terms of hyperparameter tuning and convergence analysis, display expected strong correlations between physically tightly coupled kinetic parameters, and thus establish compatibility with past work. The behavior

of our MCMC chains varied significantly across the dimensions of our kinetic parameters, and correlated with the half context frequency in our truncated state spaces, suggesting that some half contexts are underspecified. Hence, it would be worth extending the dataset with new reaction types contributing to the transition classes of currently low frequency, as well as developing kinetic models that partition the transitions into more fine-grained equivalence classes. Parameters for transition contexts which are prevalent in the energy barrier regions, e.g., as suggested by the trajectory analyses in Section 5 for hybridization, can be expected to be particularly influential for the overall kinetics. Another important finding is that the posteriors obtained for the AP and PE targets are visibly different in Figure 1, but still produce remarkably similar predictive distributions over reaction rates in Figure 2. This suggests that the posterior approximations are concentrated towards a parameter region in which the rate predictions are more strongly influenced by the state space approximation than by the kinetic parameters. It would therefore be worth improving the likelihood approximation method, such that the forward model dynamically regenerates the state space with different parameters, rather than using a constant state space truncation.

Our results also reveal severe, previously undocumented numerical instability in the current likelihood model, which predicts the MFPT by solving an often ill-conditioned linear system. Effectively, this is a truncation-dependent and solver-dependent likelihood term which penalises parts of the parameter space, influencing in a nontrivial way both the exact posterior and its numerical approximation. In principle, this issue could be mitigated by a suitably preconditioned iterative solver. However, this ill-conditioning is intrinsic for metastable systems, and problem-specific preconditioners have not yet been developed, beyond rescaling by the equilibrium distribution. Furthermore, the MFPT observation model might in general be too low-dimensional and even misspecified with respect to various regression methods used to extract reaction rates from experimental measurements.

Despite these challenges, we were able to use our new Arrhenius parameters to suggest why rates of helix association reactions vary by two orders of magnitude, even when the interacting strands have little or no stable intra-strand secondary structure. This shows the potential of elementary step models for gaining insight into the kinetic behavior of nucleic acid reactions, once properly calibrated and augmented with effective tools for truncation and coarse-graining. In future work, it would be valuable to implement a simulator that allows mathematically separate model components (e.g., experimental conditions, state spaces, initial distributions, final regions, thermodynamic and kinetic models) to be defined and parameterised independently, that supports flexible parallelism, and that provides enhanced path sampling. With such capabilities, the elementary step model could become a standard tool for designing sequences in a way that accounts for transient behavior, marking a significant improvement over thermodynamic sequence design techniques.

References

1 Victor a Bloomfield, Donald M. Crothers, and Ignacio Tinoco. *Nucleic Acids: Structures, Properties and Functions*. University Science Books, Sausalito, Calif, 2000.

2 Daniel P. Aalberts, John M. Parman, and Noel L. Goddard. Single-strand stacking free energy from DNA beacon kinetics. *Biophysical Journal*, 84(5):3212–3217, 2003. doi:10.1016/S0006-3495(03)70045-9.

3 Grégoire Altan-Bonnet, Albert J. Libchaber, and Oleg Krichevsky. Bubble dynamics in double-stranded DNA. *Physical Review Letters*, 90(13):138101, 2003. doi:10.1103/PhysRevLett.90.138101.

4 Yaniv Amir, Eldad Ben-Ishay, Daniel Levner, Shmulik Ittah, Almogit Abu-Horowitz, and Ido Bachelet. Universal computing by DNA origami robots in a living animal. *Nature Nanotechnology*, 9(5):353–357, 2014. doi:10.1038/nnano.2014.58.

5 Jonathan Bath, Simon J. Green, and Andrew J. Turberfield. A free-running DNA motor powered by a nicking enzyme. *Angewandte Chemie International Edition*, 44(28):4358–4361, 2005. doi:10.1002/anie.200501262.

6 Jonathan Bath and Andrew J. Turberfield. DNA nanomachines. *Nature Nanotechnology*, 2(5):275–284, 2007. doi:10.1038/nnano.2007.104.

7 Michael Betancourt. A short review of ergodicity and convergence of Markov chain Monte Carlo estimators. *arXiv e-prints*, 2021. doi:10.48550/arXiv.2110.07032.

8 Grégoire Bonnet. *Dynamics of DNA Breathing and Folding for Molecular Recognition and Computation*. PhD Thesis, Rockefeller University, 2000.

9 Grégoire Bonnet, Oleg Krichevsky, and Albert Libchaber. Kinetics of conformational fluctuations in DNA hairpin-loops. *Proceedings of the National Academy of Sciences*, 95(15):8602–8606, 1998. doi:10.1073/pnas.95.15.8602.

10 George E. P. Box. Sampling and Bayes' inference in scientific modelling and robustness. *Journal of the Royal Statistical Society: Series A (General)*, 143(4):383–404, 1980. doi:10.2307/2982063.

11 Ronald R. Breaker and Gerald F. Joyce. The expanding view of RNA and DNA function. *Chemistry & Biology*, 21(9):1059–1065, 2014. doi:10.1016/j.chembiol.2014.07.008.

12 Thomas R. Cech and Joan A. Steitz. The noncoding RNA revolution: Trashing old rules to forge new ones. *Cell*, 157(1):77–94, 2014. doi:10.1016/j.cell.2014.03.008.

13 Arkadiusz Chworos, Isil Severcan, Alexey Y. Koyfman, Patrick Weinkam, Emin Oroudjev, Helen G. Hansma, and Luc Jaeger. Building programmable jigsaw puzzles with RNA. *Science*, 306(5704):2068–2072, 2004. doi:10.1126/science.1104686.

14 Ibrahim I. Cisse, Hajin Kim, and Taekjip Ha. A rule of seven in Watson-Crick base-pairing of mismatched sequences. *Nature Structural & Moleuclar Biology*, 19(6):623, 2012. doi:10.1038/nsmb.2294.

15 Nadine L. Dabby. *Synthetic Molecular Machines for Active Self-Assembly: Prototype Algorithms, Designs, and Experimental Study*. PhD Thesis, California Institute of Technology, 2013. doi:10.7907/T0ZG-PA07.

16 Timothy A. Davis. Algorithm 832: UMFPACK v4.3: An unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, 2004. doi:10.1145/992200.992206.

17 Aesara Developers. Aesara. https://aesara.readthedocs.io/en/latest/. Accessed: 2023-04-10.

18 P. Diaconis and L. Saloff-Coste. What do we know about the Metropolis algorithm? *Journal of Computer and System Sciences*, 57(1):20–36, 1998. doi:10.1006/jcss.1998.1576.

19 Hendrik Dietz, Shawn M. Douglas, and William M. Shih. Folding DNA into twisted and curved nanoscale shapes. *Science*, 325(5941):725–730, 2009. doi:10.1126/science.1174251.

20 Shawn M. Douglas, Ido Bachelet, and George M. Church. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335(6070):831–834, 2012. doi:10.1126/science.1214081.

21 Shawn M. Douglas, Hendrik Dietz, Tim Liedl, Bjorn Hogberg, Franziska Graf, and William M. Shih. Self-assembly of DNA into nanoscale three-dimensional shapes. *Nature*, 459(7245):414–418, 2009. doi:10.1038/nature08016.

22 Eric C Dykeman. An implementation of the Gillespie algorithm for RNA kinetics with logarithmic time update. *Nucleic Acids Research*, 43(12):5708–5715, 2015. doi:10.1093/nar/gkv480.

23 Christoph Flamm, Walter Fontana, Ivo L. Hofacker, and Peter Schuster. RNA folding at elementary step resolution. *RNA*, 6(03):325–338, 2000. doi:10.1017/S1355838200992161.

24 Daniel Foreman-Mackey, David W. Hogg, Dustin Lang, and Jonathan Goodman. emcee: The MCMC hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306, 2013. doi:10.1086/670067.

25 Mark E. Fornace. *Computational Methods for Simulating and Parameterizing Nucleic Acid Secondary Structure Thermodynamics and Kinetics*. PhD Thesis, California Institute of Technology, 2022. doi:10.7907/ayeg-at42.

26 Jinglin Fu, Yuhe Renee Yang, Alexander Johnson-Buck, Minghui Liu, Yan Liu, Nils G. Walter, Neal W. Woodbury, and Hao Yan. Multi-enzyme complexes on DNA scaffolds capable of substrate channelling with an artificial swinging arm. *Nature Nanotechnology*, 9(7):531–536, 2014. doi:10.1038/nnano.2014.100.

27 Yang Gao, Lauren K. Wolf, and Rosina M. Georgiadis. Secondary structure effects on DNA hybridization kinetics: a solution versus surface comparison. *Nucleic Acids Research*, 34(11):3370–3377, 2006. doi:10.1093/nar/gkl422.

28 Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C. Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. Bayesian workflow. *arXiv e-prints*, 2020. doi:10.48550/arXiv.2011.01808.

29 Charles Geyer. Introduction to Markov chain Monte Carlo. In Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng, editors, *Handbook of Markov Chain Monte Carlo*, volume 20116022. Chapman and Hall/CRC, 2011. doi:10.1201/b10905.

30 Charles J. Geyer. Practical Markov chain Monte Carlo. *Statistical Science*, 7(4):473–483, 1992. doi:10.1214/ss/1177011137.

31 Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977. doi:10.1021/j100540a008.

32 Noel L. Goddard, Grégoire Bonnet, Oleg Krichevsky, and Albert Libchaber. Sequence dependent rigidity of single stranded DNA. *Physical Review Letters*, 85(11):2400, 2000. doi:10.1103/PhysRevLett.85.2400.

33 Hiroaki Hata, Tetsuro Kitajima, and Akira Suyama. Influence of thermodynamically unfavorable secondary structures on DNA hybridization kinetics. *Nucleic Acids Research*, 46(2):782–791, 2018. doi:10.1093/nar/gkx1171.

34 Ivo L. Hofacker. Vienna RNA secondary structure server. *Nucleic Acids Research*, 31(13):3429–3431, 2003. doi:10.1093/nar/gkg599.

35 Stephan Hoyer and Joe Hamman. Xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, 5(1):10, 2017. doi:10.5334/jors.148.

36 Yonggang Ke, Luvena L. Ong, William M. Shih, and Peng Yin. Three-dimensional structures self-assembled from DNA bricks. *Science*, 338(6111):1177–1183, 2012. doi:10.1126/science.1227268.

37 Jiho Kim, Sören Doose, Hannes Neuweiler, and Markus Sauer. The initial step of DNA hairpin folding: a kinetic analysis using fluorescence correlation spectroscopy. *Nucleic Acids Research*, 34(9):2516–2527, 2006. doi:10.1093/nar/gkl221.

38 Ravin Kumar, Colin Carroll, Ari Hartikainen, and Osvaldo Martin. ArviZ a unified library for exploratory analysis of Bayesian models in Python. *Journal of Open Source Software*, 4(33):1143, 2019. doi:10.21105/joss.01143.

39 Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):e0177459, 2017. doi:10.1371/journal.pone.0177459.

40 Martin Langecker, Vera Arnaut, Thomas G. Martin, Jonathan List, Stephan Renner, Michael Mayer, Hendrik Dietz, and Friedrich C. Simmel. Synthetic lipid membrane channels formed by designed DNA nanostructures. *Science*, 338(6109):932–936, 2012. doi:10.1126/science.1225624.

41 Tim Liedl, Björn Högberg, Jessica Tytell, Donald E. Ingber, and William M. Shih. Self-assembly of three-dimensional prestressed tensegrity structures from DNA. *Nature Nanotechnology*, 5(7):520–524, 2010. doi:10.1038/nnano.2010.107.

42 Pavel Loskot, Komlan Atitey, and Lyudmila Mihaylova. Comprehensive review of models and methods for inferences in bio-chemical reaction networks. *Frontiers in Genetics*, 10, 2019. doi:10.3389/fgene.2019.00549.

43 Jordan Lovrod. Bayesian modelling of DNA secondary structure kinetics: revisiting path space approximations and posterior inference in exponentially large state spaces. MSc Thesis, University of British Columbia, 2023. doi:10.14288/1.0431312.

44 Robert R. F. Machinek, Thomas E. Ouldridge, Natalie E. C. Haley, Jonathan Bath, and Andrew J. Turberfield. Programmable energy landscapes for kinetic control of DNA strand displacement. *Nature Communications*, 5(1):5324, 2014. doi:10.1038/ncomms6324.

45 Chengde Mao, Weiqiong Sun, Zhiyong Shen, and Nadrian C. Seeman. A nanomechanical device based on the B–Z transition of DNA. *Nature*, 397(6715):144–146, 1999. doi:10.1038/16437.

46 David H. Mathews, Jeffrey Sabina, Michael Zuker, and Douglas H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *Journal of Molecular Biology*, 288(5):911–940, 1999. doi:10.1006/jmbi.1999.2700.

47 Sean A. McKinney, Alasdair D. J. Freeman, David M. J. Lilley, and Taekjip Ha. Observing spontaneous branch migration of Holliday junctions one step at a time. *Proceedings of the National Academy of Sciences*, 102(16):5715–5720, 2005. doi:10.1073/pnas.0409328102.

48 Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. doi:10.1063/1.1699114.

49 Larry E. Morrison and Lucy M. Stols. Sensitive fluorescence-based thermodynamic and kinetic measurements of DNA hybridization in solution. *Biochemistry*, 32(12):3095–3104, 1993. doi:10.1021/bi00063a022.

50 G. Eric Plum, Kenneth J. Breslauer, and Richard W. Roberts. 7.02 - Thermodynamics and kinetics of nucleic acid association/dissociation and folding processes. In *Comprehensive Natural Products Chemistry*, pages 15–53. Pergamon, Oxford, 1999. doi:10.1016/B978-0-08-091283-7.00056-4.

51 Brittany Rauzan, Elizabeth McMichael, Rachel Cave, Lesley R. Sevcik, Kara Ostrosky, Elisabeth Whitman, Rachel Stegemann, Audra L. Sinclair, Martin J. Serra, and Alice A. Deckert. Kinetics and thermodynamics of DNA, RNA, and hybrid duplex formation. *Biochemistry*, 52(5):765–772, 2013. doi:10.1021/bi3013005.

52 Luis P. Reynaldo, Alexander V. Vologodskii, Bruce P. Neri, and Victor I. Lyamichev. The kinetics of oligonucleotide replacements. *Journal of Moleuclar Biology*, 297(2):511–520, 2000. doi:10.1006/jmbi.2000.3573.

53 Christian P. Robert and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999. doi:10.1007/978-1-4757-4145-2.

54 Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006. doi:10.1038/nature04586.

55 Vivekananda Roy. Convergence diagnostics for Markov chain Monte Carlo. *Annual Review of Statistics and Its Application*, 7:387–412, 2020. doi:10.1146/annurev-statistics-031219-041300.

56 John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, 2016. doi:10.7717/peerj-cs.55.

57 John SantaLucia. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proceedings of the National Academy of Sciences*, 95(4):1460–1465, 1998. doi:10.1073/pnas.95.4.1460.

58 John SantaLucia Jr. and Donald Hicks. The thermodynamics of DNA structural motifs. *Annu. Rev. Biophys. Biomol. Struct.*, 33:415–440, 2004. doi:10.1146/annurev.biophys.32.110601.141800.

59 Joseph M. Schaeffer. *Stochastic Simulation of the Kinetics of Multiple Interacting Nucleic Acid Strands*. PhD Thesis, California Institute of Technology, 2013. doi:10.7907/JEBY-6X69.

60 Joseph M. Schaeffer, Chris Thachuk, and Erik Winfree. Stochastic simulation of the kinetics of multiple interacting nucleic acid strands. In *DNA Computing and Molecular Programming*, volume 9211 of *Lecture Notes in Computer Science*, pages 194–211, 2015. doi:10.1007/978-3-319-21999-8_13.

61 John S. Schreck, Thomas E. Ouldridge, Flavio Romano, Petr Šulc, Liam P. Shaw, Ard A. Louis, and Jonathan P.K. Doye. DNA hairpins destabilize duplexes primarily by promoting melting rather than by inhibiting hybridization. *Nucleic Acids Research*, 43(13):6181–6190, 2015. doi:10.1093/nar/gkv582.

62 Verena J. Schüller, Simon Heidegger, Nadja Sandholzer, Philipp C. Nickels, Nina A. Suhartha, Stefan Endres, Carole Bourquin, and Tim Liedl. Cellular immunostimulation by CpG-sequence-coated DNA origami structures. *ACS Nano*, 5(12):9696–9702, 2011. doi:10.1021/nn203161y.

63 Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314(5805):1585–1588, 2006. doi:10.1126/science.1132493.

64 Nadrian C. Seeman. Nucleic acid junctions and lattices. *Journal of Theoretical Biology*, 99(2):237–247, 1982. doi:10.1016/0022-5193(82)90002-9.

65 Alexander Serganov and Evgeny Nudler. A decade of riboswitches. *Cell*, 152(1-2):17–24, 2013. doi:10.1016/j.cell.2012.12.024.

66 Daniel J. Sharpe and David J. Wales. Efficient and exact sampling of transition path ensembles on Markovian networks. *The Journal of Chemical Physics*, 153(2):024121, 2020. doi:10.1063/5.0012128.

67 Daniel J. Sharpe and David J. Wales. Nearly reducible finite Markov chains: Theory and algorithms. *The Journal of Chemical Physics*, 155(14):140901, 2021. doi:10.1063/5.0060978.

68 Niranjan Srinivas, Thomas E. Ouldridge, Petr Šulc, Joseph M. Schaeffer, Bernard Yurke, Ard A. Louis, Jonathan P. K. Doye, and Erik Winfree. On the biophysics and kinetics of toehold-mediated DNA strand displacement. *Nucleic Acids Research*, 41(22):10641–10658, 2013. doi:10.1093/nar/gkt801.

69 Yuri Suhov and Mark Kelbert. *Markov Chains: A Primer in Random Processes and Their Applications*, volume 2. Cambridge University Press, 2008.

70 Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, 2016. doi:10.48550/arXiv.1605.02688.

71 Gael Varoquaux and Olivier Grisel. Joblib: Running Python functions as pipeline jobs. https://joblib.readthedocs.io/en/latest/, 2009. Accessed: 2023-04-15.

72 Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.

73 Mark I. Wallace, Liming Ying, Shankar Balasubramanian, and David Klenerman. Non-Arrhenius kinetics for the loop closure of a DNA hairpin. *Proceedings of the National Academy of Sciences*, 98(10):5584–5589, 2001. doi:10.1073/pnas.101523498.

74 Anthony S. Walsh, HaiFang Yin, Christoph M. Erben, Matthew J. A. Wood, and Andrew J. Turberfield. DNA cage delivery to mammalian cells. *ACS Nano*, 5(7):5427–5432, 2011. doi:10.1021/nn2005574.

75 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD Thesis, California Institute of Technology, 1998. doi:10.7907/HBBV-PF79.

76 Joseph N. Zadeh, Conrad D. Steenberg, Justin S. Bois, Brian R. Wolfe, Marshall B. Pierce, Asif R. Khan, Robert M. Dirks, and Niles A. Pierce. NUPACK: Analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, 32(1):170–173, 2011. doi:10.1002/jcc.21596.

77 David Yu Zhang and Erik Winfree. Control of DNA strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society*, 131(47):17303–17314, 2009. doi:10.1021/ja906987s.

78 Jinny X. Zhang, John Z. Fang, Wei Duan, Lucia R. Wu, Angela W. Zhang, Neil Dalchau, Boyan Yordanov, Rasmus Petersen, Andrew Phillips, and David Yu Zhang. Predicting DNA hybridization kinetics from sequence. *Nature Chemistry*, 10(1):91–98, 2018. doi:10.1038/nchem.2877.

79 Jinny X. Zhang, Boyan Yordanov, Alexander Gaunt, Michael X. Wang, Peng Dai, Yuan-Jyue Chen, Kerou Zhang, John Z. Fang, Neil Dalchau, Jiaming Li, Andrew Phillips, and David Yu Zhang. A deep learning model for predicting next-generation sequencing depth from DNA sequence. *Nature Communications*, 12(1):4387, 2021. doi:10.1038/s41467-021-24497-8.

80 Yong-Xing Zhao, Alan Shaw, Xianghui Zeng, Erik Benson, Andreas M. Nyström, and Björn Högberg. DNA origami delivery system for cancer therapy with tunable release properties. *ACS Nano*, 6(10):8684–8691, 2012. doi:10.1021/nn3022662.

81 Sedigheh Zolaktaf. *Efficiently estimating kinetics of interacting nucleic acid strands modeled as continuous-time Markov chains*. PhD Thesis, University of British Columbia, 2020. doi:10.14288/1.0395346.

82 Sedigheh Zolaktaf, Frits Dannenberg, Xander Rudelis, Anne Condon, Joseph M. Schaeffer, Mark Schmidt, Chris Thachuk, and Erik Winfree. Inferring parameters for an elementary step model of DNA structure kinetics with locally context-dependent Arrhenius rates. In *DNA Computing and Molecular Programming*, volume 10467 of *Lecture Notes in Computer Science*, pages 172–187, 2017. doi:10.1007/978-3-319-66799-7_12.

83 Sedigheh Zolaktaf, Frits Dannenberg, Mark Schmidt, Anne Condon, and Erik Winfree. Predicting DNA kinetics with a truncated continuous-time Markov chain method. *Computational Biology and Chemistry*, 104:107837, 2023. doi:10.1016/j.compbiolchem.2023.107837.

84 Sedigheh Zolaktaf, Frits Dannenberg, Erik Winfree, Alexandre Bouchard-Côté, Mark Schmidt, and Anne Condon. Efficient parameter estimation for DNA kinetics modeled as continuous-time Markov chains. In *DNA Computing and Molecular Programming*, volume 11648 of *Lecture Notes in Computer Science*, pages 80–99, 2019. doi:10.1007/978-3-030-26807-7_5.

85 Michael Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Research*, 31(13):3406–3415, 2003. doi:10.1093/nar/gkg595.

A Appendix

A.1 Criteria for initial and final regions of a reaction

A.2 PE method

The PE algorithm has five hyperparameters, b, κ, n_b, n_κ, and δ (referred to as β, κ, N, K, and δ in the original reference [83]). For a CTMC with transition rate matrix K, probability matrix P and final region F, let $\mathbb{1}_{\text{dist}} : \mathcal{X}^2 \to \{0, 1\}$ be the decreasing-distance indicator that maps adjacent states x and x' to 1 if the minimum distance (in steps) from x' to any absorbing state $x_f \in F$ is less than the distance from x to x_f, and to 0 otherwise. This indicator is used to define P_{bias}, which alters P by only allowing for transitions that decrease the distance to the final region, and P_b for any $b \in [0, 1]$ is taken to be the convex combination of P_{bias} and P:

$$P_{\text{bias}}(x, x') = \frac{K(x, x')\mathbb{1}_{\text{dist}}(x, x')}{\sum_{x'' \in \mathcal{X}} K(x, x'')\mathbb{1}_{\text{dist}}(x, x'')}\,, \tag{11}$$

$$P_b = bP + (1 - b)P_{\text{bias}}\,. \tag{12}$$

For a given P, we refer to samples from P_b as b-biased. The PE algorithm can be summarised in four steps:

Table 1 Criteria for the initial and final regions. Note that, although some of these definitions allow for endpoint regions containing several secondary structure states, the state spaces constructed by the AP approximation only include one state satisfying each criterion.

Bubble closing	*Initial:* The microstate where all bases in the scope of the specified bubble are unpaired, and all other bases are paired as in the final state *Final:* The microstate with the fully formed hairpin
Hairpin opening	*Initial:* The microstate with the fully formed hairpin *Final:* The microstate with no base pairs
Hairpin closing	*Reverse of hairpin opening*
Helix association	*Initial:* Any microstate with no base pairs between strands *Final:* The microstate with the fully formed helix
Helix dissociation	*Reverse of helix association*
Strand displacement	*Initial:* Any microstate where the incumbent and substrate form a complex without the invader *Final:* Any microstate where the invader and substrate form a complex without the incumbent

1. **Pathway construction.** Sample n_b distance-biased trajectories from initial region I to final region F, such that the holding times are sampled according to the diagonal entries of K, while transitions are sampled according to P_b.
2. **State elaboration.** For each state x discovered during the construction step, sample n_κ unbiaed paths according to K, starting at x with time limit κ.
3. **Transition construction.** Construct a new rate matrix $\hat{K}$ such that for any states x and x' discovered during simulations, $\hat{K}(x, x') := K(x, x')$.
4. **δ-pruning.** Group all states that are within a fixed MFPT δ of any $x_f \in F$ into a single absorbing state, and update the rate matrix $\hat{K}$ accordingly.

The hyperparameters κ and b directly improve the approximation quality of the truncated CTMC, while n_κ and n_b only increase the approximation quality if κ and b are suitable for the CTMC in question. In brief, increases in b decrease the degree of bias in the pathway construction step, increases in κ increase the expected length of each unbiased trajectory sample, and increases in n_b/n_κ increase the number of biased/unbiased trajectory samples, respectively. In particular, when $b = 1$ and $n_\kappa = 0$, SSA is recovered as a special case. In practice, the current implementation of PE is inefficient in time and memory usage, which strongly limits the practically achievable hyperparameters and approximation quality.

A.3 PE hyperparameter value sets

Table 2 Sets of PE hyperparameter values attempted on each reaction. The ordering of the sets of values corresponds to decreasing expected state space size. This order was also used to prioritise which CTMCs to include in our inference targets (Appendix Table 4), for reactions where more than one set of hyperparameters produced a valid CTMC.

	P1B1	**P1B2**	**P2B1**	**P2B2**	**P3B1**	**P3B2**	**P4B1**	**P4B2**
n_b	128	128	64	64	32	32	16	16
n_κ	256	256	128	128	64	64	32	32
b	0.4	0.1	0.4	0.1	0.4	0.1	0.4	0.1
κ	16ns	16ns	16ns	16ns	16ns	16ns	16ns	16ns

A.4 Dataset

Table 3 Dataset of experimentally measured rate coefficients. The sign * next to the number of reactions indicates that the dataset includes mismatch experiments.

Reaction type	no. reactions	no. bases	°C	$\log_{10}\left(k\,[M^{-1}s^{-1}]\right)$	ref.
Bubble closing	18	62	21.9 – 48.7	3.8 – 4.5	[3]
Hairpin opening	63	22 – 40	10.3 – 48.8	1.4 – 4.6	[9]
	79	20 – 40	17.8 – 48.7	2.2 – 4.7	[8, 32]
	8	40	9.5 – 45.6	0.9 – 3.1	[73]
	22	8	9.9 – 60.5	4.6 – 6.0	[37]
Hairpin closing	62	22 – 40	10.0 – 49.4	3.4 – 4.8	[9]
	102	18 – 40	14.3 – 48.8	2.8 – 5.3	[8]
	8	40	9.8 – 45.6	3.2 – 3.4	[73]
	22	8	9.9 – 60.6	4.6 – 6.1	[37]
	27	31	14.1 – 41.1	2.7 – 4.0	[2]
Helix association	15	20 – 40	3.4 – 49.3	5.9 – 7.6	[49]
	47	46	25.0	4.0 – 6.7	[33]
	210	72	28.0 – 55.0	4.2 – 7.4	[78]
	39*	18 – 20	23.0 – 37.0	4.2 – 7.4	[14]
	9	50	23.0	4.9 – 6.2	[27]
	18	16	6.6 – 33.6	6.5 – 7.3	[51]
Helix dissociation	12	20 – 40	24.7 – 68.0	-2.7 – -1.0	[49]
	14	42 – 46	30.0 – 55.0	-5.3 – -2.9	[52]
	39*	18 – 20	23.0 – 37.0	-1.2 – 0.9	[14]
Strand displacement	30	78 – 96	25.0	0.9 – 7.0	[77]
	14	54 – 62	30.0 – 55.0	0.6 – 1.9	[52]
	36*	83 – 87	23.0	2.7 – 6.8	[44]
	211	89 – 102	28.0 – 55.0	1.3 – 8.2	[79]
Overall	1105	8 – 102	3.4 – 68.0	-5.3 – 8.2	

A.5 Inference targets

Table 4 State spaces used in each inference target.

	no. reactions	avg. no. states
AP target		
Bubble closing	18	758
Hairpin opening	221	46
Hairpin closing	172	44
Helix association	338	1843
Helix dissociation	65	275
Strand displacement	291	9626
Total	1105	3143
PE target		
Bubble closing	18	2048
Hairpin opening	221	3120
Hairpin closing	172	1268
Helix association	163	26113
Helix dissociation	65	23223
Strand displacement	44	27045
Total	683	11567

A.6 MCMC trace plots

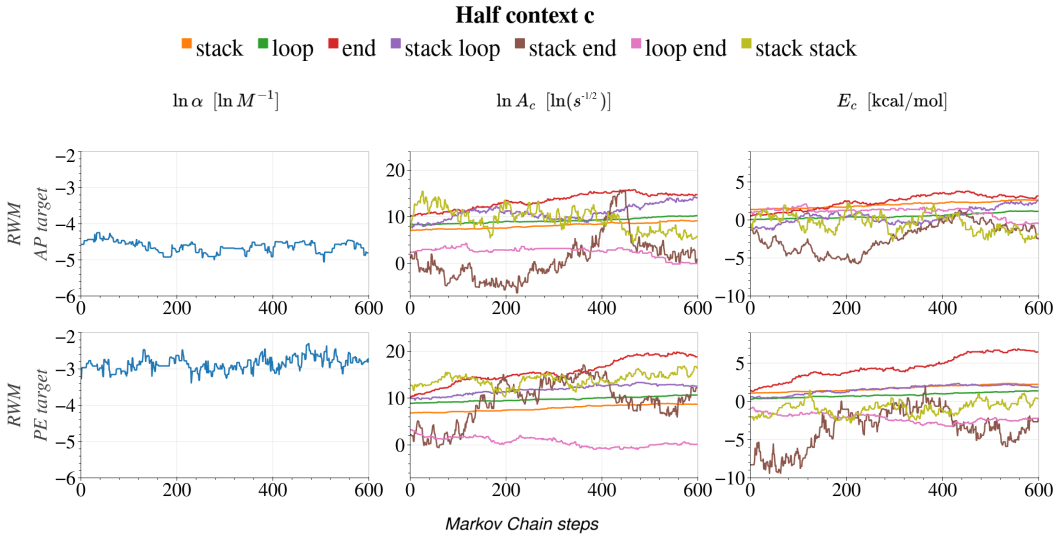


Figure 7 Trace plots from the RWM sampler for AP and PE inference targets, corresponding to Figure 1. Burn-in samples are not shown.

Reversible Bond Logic

Hannah Amelie Earley
Department of Applied Mathematics and Theoretical Physics, University of Cambridge, UK

Abstract

The field of molecular programming allows for the programming of the structure and behavior of matter at the molecular level, even to the point of encoding arbitrary computation. However, current approaches tend to be wasteful in terms of monomers, gate complexes, and free energy. In response, we present a novel abstract model of molecular programming, Reversible Bond Logic (RBL), which exploits the concepts of reversibility and reversible computing to help address these issues. RBL systems permit very general manipulations of arbitrarily complex "molecular" structures, and possess properties such as component reuse, modularity, compositionality. We will demonstrate the implementation of a common free-energy currency that can be shared across systems, initially using it to power a biased walker. Then we will introduce some basic motifs for the manipulation of structures, which will be used to implement such computational primitives as conditional branching, looping, and subroutines. Example programs will include logical negation, and addition and squaring of arbitrarily large numbers. As a consequence of reversibility, we will also obtain the inverse programs (subtraction and square-rooting) for free. Due to modularity, multiple instances of these computations can occur in parallel without cross-talk. Future work aims to further characterize RBL, and develop variants that may be amenable to experimental implementation.

2012 ACM Subject Classification Computer systems organization → Molecular computing; Theory of computation → Models of computation

Keywords and phrases Molecular Programming, Reversible Computing, Structural Manipulation

Digital Object Identifier 10.4230/LIPIcs.DNA.29.6

Acknowledgements The author would like to thank Gos Micklem, Jim Lathrop, William Poole, and three anonymous reviewers for their valuable comments and suggestions.

1 Introduction

Molecular programming is the powerful idea that we can program the very structure and behavior of matter at the molecular level. Not only can we build nanostructures of nearly arbitrary shape and functional properties, but we can design molecules whose interactions encode computation. Perhaps the earliest known example was able to compute solutions to NP-complete problems, in particular the Hamiltonian path problem [1]. Since then, implementations of models such as the Tile Assembly Model (TAM) and finite Chemical Reaction Networks (CRNs) have arisen [32, 28], leading to the possibility of general Turing-universal computation [27].

Versatile as these schemes are, for the most part our approaches to molecular computation are wasteful. For one, they tend to be one-shot [1, 23, 35]. Computations are set up in a state far from equilibrium, with free energy dispersed throughout the various species (typically complexes of DNA). Computation then typically corresponds to the process of equilibration of this system [26]. While this certainly provides a high driving force for the experiment, it has a number of drawbacks. In the case of many DNA-strand displacement systems, which typically consist of signal strands and gate complexes, many of the components cannot be reused; rather they are transformed into a plethora of waste complexes [35]. As the variety of waste complexes is large, it is non-trivial to selectively remove them and replace them with fresh gate complexes. Consequently, the runtime of computation is generally constrained by the initial concentrations of species. Moreover, the prospect of a long-running computational

system that responds to changing inputs over time – a feature that may be desirable in a smart therapeutic, for example – becomes impractical at best. Nevertheless, some work on renewable or long-running dynamic DNA-based systems is beginning to emerge [7, 10], as well as systems that reuse components [6]. As for Tile Assembly systems, free tile monomers are typically locked into a final assembly (except for transiently at the growing edge) [11]. Not only does this eventually starve future computation, but often these consumed tiles serve no ongoing functional purpose save that of storing the history of computation. Signal-passing Tile Assembly models [22] are capable of modifying the state of tiles after incorporation, such as to remove them, but these are irreversible changes.

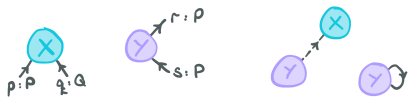
Is this waste – of free energy, monomers, and special complexes – unavoidable? Turning to life, the maestra of molecular machines, we see that this is not the case. Biochemical systems routinely recycle monomers – building up and breaking down macromolecules from their constituent components. Their molecular machines (enzymes), which we loosely identify with the gate complexes mentioned earlier, are in general fully reusable – acting catalytically. Lastly, free energy is not distributed casually across a large number of components. Rather, a select few species are designated as free energy currency. The prototypical example is that of ATP/ADP. This pair of chemical species can be converted between one another, so by maintaining a ratio of the two that is far from equilibrium, the hydrolysis of ATP into ADP can be used to store free energy. The missing ingredient is then to build molecular machines that couple a desired reaction to this hydrolysis, so the free energy can be supplied to other reactions. In this way, living systems need only inject fresh free energy into these few subsystems in order to continually sustain the operation of all other processes.

From this we draw one main conclusion for the design of more effective molecular systems. Free energy should be separated out from the molecular machines it powers. As a consequence, the molecular machines will generally be catalytic, returning to their original state after performing their function. This is best implemented by exploiting reversible dynamics, which is already a characteristic of the microscopic realm. In reversible dynamics, the previous state of a system (and indeed, its entire history) is uniquely determined by the current state. More strongly, we have time reversal symmetry: the laws of physics are the same both forwards and backwards. An interesting corollary of these reversible dynamics is that the system's evolution (at least at the local level) can be reversed by inverting the free energy supply. Additionally, the speed of evolution can be controlled by increasing or decreasing the amount of free energy stored. Of course, these are not unknown ideas to the molecular programming community. One of the earliest proposals for a molecular computer, arguably predating the field, is Bennett's enzymatic Turing machine [3]. Bennett's design consisted of a polymeric tape, bespoke enzymes performing reversible computational steps, and a pool of free energy currency and structural monomers. While not experimentally realized, other enzymatic approaches to molecular computation have been, such as the PEN toolbox [21] and PER [17]. As designing custom enzymes remains highly non-trivial, however, these approaches reuse naturally occurring enzymes such as DNA polymerase. Non-enzymatic approaches include the proposed DNA polymer stack machines of Qian et al. [23], the reversible surface CRNs of Brailovskaya et al. [4], and intricate chemomechanical systems [5, 24]. Furthermore, reversibility often features as a building block in other systems [12, 14, 16].

We present a novel abstract model of molecular programming, Reversible Bond Logic (RBL). RBL systems consist of "atoms", which can be combined by "bonds" into arbitrarily complex "molecules". By programming the energy landscape of the atom-bond configurations, reversible paths through configuration space can be carved. This will prove sufficient to implement a variety of systems, including catalytic molecular machines and a common free

$\mathcal{S} = (\{X, Y\}, \{P, Q\}, \{\text{solid}, \text{dashed}\}, \mathbb{R}_{\geq 0} \cup \{\infty\})$
$P = \{\text{solid}, \text{dashed}\} \qquad Q = \{\text{solid}\}$
$X = (\{p, q\}, (p : P, q : Q), (p : \text{in}, q : \text{in}), e_X)$
$Y = (\{r, s\}, (r : P, s : P), (r : \text{out}, s : \text{in}), e_Y)$

(a)



(b) **(c)**

Figure 1 An example RBL scheme, $\mathcal{S}$. **(a)** The formal definition of $\mathcal{S}$, which comprises two atom types, X and Y, two port types, P and Q, and two bond colors, solid and dashed. Energies may be any non-negative real or infinite. Each atom has two ports, and the energy configurations (e_X, e_Y) are left unspecified. **(b)** The RBL atoms defined in diagrammatic form. The label $p : P$ indicates that the port has label p and is of type P. **(c)** An example system configuration with one copy of X and two of Y. As we keep the same port positions as in (b), we omit port labels for brevity.

energy currency. Moreover, as RBL atoms are manipulated in a reversible fashion, they can be freely reused. Indeed, in RBL complex and diverse macromolecular structures can be built up and broken down as required. The informational content of these structures can then be exploited to build modular and compositional computational entities, allowing for rich computational primitives such as looping, recursion, and subroutines. The goal of RBL is to be a new platform for molecular programming, in particular it is hoped that it will enable the routine manipulation of complex structures and that it will make the design of energy efficient systems easier. As a corollary, RBL may also provide a new route for the implementation and study of reversible computing.

It is important to note that reversibility has profound implications on the very nature of computation, and so will affect how are our programs are written. Conventional approaches to computation make liberal use of non-invertible computational primitives, such as overwriting variables or (freely) merging branches of control flow. The most immediate consequence is that it is not generally possible to determine the previous state of a computer from the current state, as there are often many possible consistent histories. While beyond the scope of this paper, there is a deep and rich connection between irreversibility in computation and the thermodynamics of information [29, 19]. As we seek reversible dynamics, we will leverage the principles of reversible computing [3] and programming [20, 34][9, pp. 11–23]. Namely, we will avoid loss of information and take care to distinguish branches of control flow when they are merged.

The paper begins with a definition of RBL. Next, RBL is introduced more concretely with the implementation of a walker powered by an external fuel supply. Then, one possible scheme for computation is presented; a small selection of structural-manipulation primitives are introduced, and used to implement conditional branching, looping, and subroutines. Additionally, the computation is coupled to the same fuel supply introduced earlier, in order to drive the otherwise-unbiased reversible computation forward. The paper concludes with a discussion of the advantages and limitations of RBL. Elaboration of the computational primitives introduced and the design decisions behind them is presented in the technical appendix.

2 Definition

Informally, an RBL scheme consists of a set of atoms. These atoms are decorated with ports of certain types, and like ports can form bonds between one another – whether between ports on two different atoms or on the same atom (a self-loop). Ports additionally come

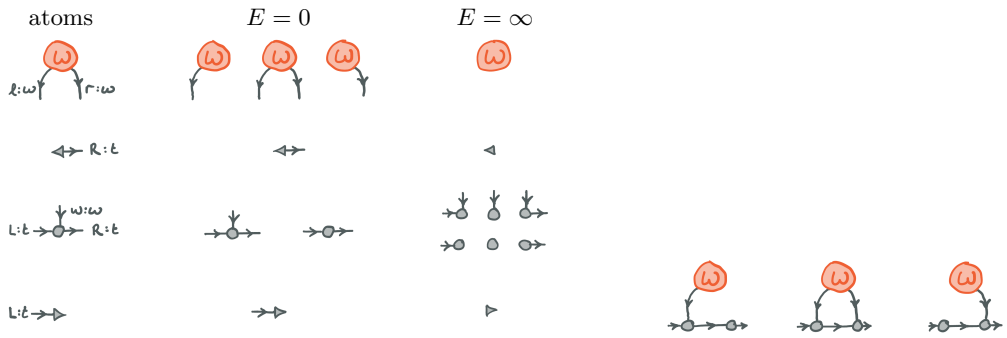
in "oriented" pairs: "in" and "out"; a bond is then formed *directionally* from an out-port to an in-port. A given port type may support bonds of multiple different "colors" (not necessarily literal colors), which can be useful for signal passing. The decoration of ports on an atom is geometry-free: there is no intrinsic ordering or positioning of ports, and they can be considered freely labile. There is some resemblance of RBL to Thermodynamic Binding Networks (TBNs) [8], which will be discussed in Section 5.

Key to RBL is the logic that dictates permissible bond formation and transitions. Each possible configuration of an atom – namely, whether each of its ports is bonded and, if so, with what color bond – is assigned an energy. As expected, higher energies are thermodynamically less likely with "$E = \infty$" representing an impossible configuration. In fact, in practice configurations are usually only assigned energies of $E = 0$ or $E = \infty$, i.e. possible or impossible, although other finite positive energies may be employed for short-lived transitional states. A system configuration – the configuration of multiple atoms – may transition to another configuration if and only if: (1) the configurations differ by a single bond (equivalently, by two port states), (2) the configuration energies are finite. That is, only one bond transition (whether formation, color change, or breakage) may occur at a given time. The kinetics may depend on the energy and associated entropy changes, e.g. due to a change in particle number, but we leave a detailed treatment of this to future work.

Designing an RBL scheme then amounts to carefully picking the energy landscapes of each atom so as to carve out a guided path(s) through configuration space. In general, we seek to restrict the system so that at any one time there are precisely two possible transitions: one to the previous state, and one to the next state. Sometimes, however, it may be desirable to provide parallel paths through configuration space when the order of operations is unimportant.

Although we will introduce RBL schemes diagramatically, we define RBL formally for completeness. An RBL scheme is specified by a tuple (A, P, B, E) where A is an indexed set of atoms, P an indexed set of ports, B a set of bond colors, and E a set of possible energies. The indices of A and P are used to uniquely label each type of atom and port. A port p consists of a subset of B, i.e. $P_p \subseteq B$, corresponding to the bond colors it can form. An atom of type α consists of a tuple $(J, \vec{p}, \vec{o}, e)$, where J is an index set enumerating the ports of the atom, $\vec{p}$ is a vector of port types indexed by J, $\vec{o}$ is a vector of port orientations indexed by J, and e is an energy function. An atom's energy function maps each possible configuration to its energy, i.e. $e : \prod_{\ell \in J}(\{\varepsilon\} \cup P_{p_\ell}) \to E$ where ε corresponds to an unbound port. An example scheme is shown in Figure 1.

A system configuration is a directed graph. Each atom of type α, with atom-tuple $(J, \vec{p}, \vec{o}, e)$, has an associated sub-graph. This sub-graph consists of an "atom" node labeled α and a set of "port" nodes: for each $j \in J$ we add an edge labeled j from the atom-node to the port-node, which is labeled (p_j, o_j). A configuration consists of a (disjoint) union of atom-graphs, where there may be any number $\in \mathbb{N}$ of copies of each type of atom-graph. Bonds correspond to edges from a node (p, out) to a node (p, in) for some port type p, and are labeled with some color in P_p. For each atom-sub-graph, we can compute its energy using the energy function e. The system configuration is valid if each of the energies of its constituent atoms is finite, and the total energy of the system is given by the sum of these. Two system configurations are *adjacent* if they are both valid and they differ by a single port-port edge; this difference can be the presence/absence of such an edge, or a change in label. A system may transition to any adjacent configuration.



(a) The RBL energy configuration of an unbiased walker.

(b) The dynamics of walking.

Figure 2 An RBL implementation of an unbiased walker. **(a)** The system consists of four types of atom. W represents the walker; it has two feet, given by the left (ℓ) and right (r) ports, which are both of type w and can form bonds of a single color, solid. The other atoms represent the track, and comprise a left-cap, track monomers, and a right-cap. The track monomers have a w port of type w, allowing the feet of the walker to bind to them. The other ports, L and R of type t (monochromatic solid), allow the track monomers to polymerize; if capped on the ends they will form a linear track, else a circular loop. The energy configuration is defined by assigning each possible state to $E = 0$ (allowed) or $E = \infty$ (impossible). The energy configuration of W allows a walker to bind to one or two track monomers, but it can never dissociate from the track as the unbound state is impossible. Meanwhile, the energy configuration of the track enforces that it must be complete and cannot fall apart. Optionally, the "transition" state where W binds to two track monomers can be assigned a higher energy, e.g. $E = 1$. **(b)** The energy configuration leads to the walking dynamics shown. The walker is free to step to the left or right, forming a transient state with two feet on the track. Then either foot can dissociate, possibly leading to net movement along the track.

3 Walkers and Fuel

A walker is a molecular machine that walks along a molecular track. In nature these are often referred to as motor proteins, and serve a vital role in cells performing such tasks as transporting cargo or contracting muscle cells. Molecular programmers have designed many instances of walkers. Some walkers have no requirement of directional movement and perform a random walk through their domain, such as the "cargo-sorting robot" of Thubagere et al. [30], or directionality is provided by cycling reagents [25]. Other walkers, such as that of Yin et al. [33], achieve uni-directional movement along a one dimensional track by using high-free energy fuel to permanently block previously-visited track footholds. This can be considered analogous to a Brownian ratchet. Such "burnt bridge" approaches to walking have the disadvantage that it is not possible to send multiple walkers down the same track, and the track needs to be rebuilt in order to walk along it again. Motor proteins do not modify their track. Instead, processive movement is conferred by supply of an external free energy currency. We will implement such a walker within RBL, starting with an unbiased walker that has no directional preference and then adding both polarity and an external fuel supply that couples directional movement to the free energy supply. While this is not the first directional walker that leaves its track intact [13], its implementation is particularly natural and yields a common free energy currency.

3.1 Unbiased Walker

An unbiased walker is relatively easy to implement, and an appropriate RBL scheme is shown in Figure 2. The walker itself is given by a single RBL atom, W, and it walks along a track using its two "feet" ports to make bonds as needed. All that is required for correct operation is for W to be able to bind to the track with one or both feet, but not none. In this way, it can make "progress" by putting down a second foot and lifting the first foot. In both the unbiased and biased walkers, detachment from the track is made impossible by assigning infinite energy to this state; in a realistic system, it may be desirable to permit controlled attachment and detachment. The techniques used in Section 4 and beyond could be used to do this. As the dynamics are reversible and history-free, there is of course no guarantee that it won't just lift the second foot and make no progress, nor is there any guarantee that it will make net progress over time. Indeed, the statistics of movement are described by an unbiased random walk with zero mean displacement after t steps, and $\sim \mathcal{O}(\sqrt{t})$ variance.

A brief commentary on the geometry of the walker atom W is warranted. If the ports of W had fixed position, then there might be a risk of the walker not moving beyond the initial pair of track monomers. Free rotation about the bonds or sufficient flexibility would fix this, however recall from the definition of RBL that the ports have no intrinsic position and are freely labile. As such, the ports are free to "move" as needed. A further consequence of being geometry-free is that a walker may "skip" an arbitrary number of positions along the track; there are a number of ways to address this, but we defer discussion of this to future work which will introduce a geometric model for RBL.

Also of note is the importance of the directionality of the bonds. Without directionality, the w ports of the track monomers would be able to bind to each other, as would the ℓ and r feet of the walker W. Therefore directionality enforces a certain complementarity relation, which is not unfamiliar in the world of DNA nanostructures.

3.2 Fuel

In order to build a powered directional walker, we will need an external fuel supply. Inspired by biochemistry, we choose to store our free energy in the population of two related species. For simplicity, we design (Figure 3) a bistable atom G with two distinct states, G_+ and G_-, that can be interconverted by an external signal at the coupling port f. Consequently, the equilibrium state corresponds to $[\mathsf{G}_+] = [\mathsf{G}_-]$ and free energy may be stored by increasing the ratio $[\mathsf{G}_+] : [\mathsf{G}_-]$ above 1. Specifically, the amount of free energy is given simply by $kT \log([\mathsf{G}_+]/[\mathsf{G}_-])$. For another system X to usefully access this free energy, it needs to couple carefully to the fuel species. The initial state of the system $X_{\text{init.}}$ must couple to the G_+ state, and the final state of the system $X_{\text{fin.}}$ must couple to the G_- state. That is, we need to implement the "reaction" $X_{\text{init.}} + \mathsf{G}_+ \rightleftharpoons X_{\text{fin.}} + \mathsf{G}_-$. This is the purpose of the colorings available on the f port, and will be shown more concretely in the next subsection.

Critical to the design is the double self-loop on G. A single self-loop would be able to autonomously transition between bond colors due to the rules of RBL, but a double self-loop allows us to impose an (infinite) energy barrier between the G_+ and G_- states. This barrier is lowered by the action of the coupling port, f, which thus acts as a catalyst for the interconversion of states.

3.3 Biased Walker

To construct a walker with a preferred walking direction, we must introduce polarity into both the track and the walker. In nature, a single monomer type (e.g. actin) is sufficient for this by taking advantage of its spatial substructure. In RBL, however, such spatial

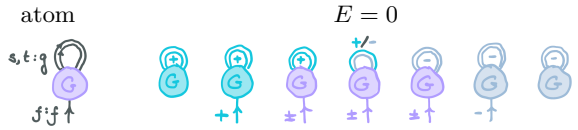


Figure 3 An RBL implementation of a bistable fuel species, G. The atom has five ports, s_1/s_2, t_1/t_2, and f. The ports s_1/s_2 form a self-loop, and so we refer to them together as s; similarly for the self-loop t. The loops are each of type g, which has two colors: $+$ (bright blue) and $-$ (gray-blue). The "coupling" port f is of type f and has three colors: $+$, $\pm$ (lilac), and $-$. The two stable states of G correspond to both loops being $+$ (G_+) or both being $-$ (G_-). These can be interconverted via the coupling port, f, passing through a transitional state $\mathsf{G}_\pm$ where s is colored $+$ and t is colored $-$. The $+$ and $-$ colorings allow another RBL atom to distinguish G_+ from G_-, whilst the $\pm$ coloring allows interconversion. If there is more G_+ than G_- then there will be a negative (favorable) free energy change ΔG associated with the reaction $\mathsf{G}_+ \to \mathsf{G}_-$. Note that we only show the possible states ($E = 0$); all other configurations may be presumed impossible ($E = \infty$).

substructure is non-existent. Instead we employ an alternating sequence of monomers. Two monomers, i.e. a sequence $\cdots\alpha\beta\alpha\beta\cdots$, would be insufficient to distinguish forward from backward movement; the minimum sequence to introduce polarity consists of three monomers, i.e. $\cdots\alpha\beta\gamma\alpha\beta\gamma\cdots$. With such a polarized track, the walker can then be modified such that, from an initial position on monomer α, the forward step takes it to β and the backward step to γ. Similar behavior applies to the other track positions. Of course, without a fuel supply forward and backward steps are identical and so movement is still non-directional. To complete the implementation, we couple these steps to consumption of fuel as follows:

$$\mathsf{W}_\alpha + \mathsf{G}_+ \rightleftharpoons \mathsf{W}_\beta + \mathsf{G}_- \qquad \mathsf{W}_\beta + \mathsf{G}_+ \rightleftharpoons \mathsf{W}_\gamma + \mathsf{G}_- \qquad \mathsf{W}_\gamma + \mathsf{G}_+ \rightleftharpoons \mathsf{W}_\alpha + \mathsf{G}_- \tag{1}$$

There are many possible designs, but we choose a three-footed walker with a fuel coupling port f. Each foot is specific to a particular monomer type, i.e. α, β, or γ, and the dynamics of "walking" resemble a wheel rolling along the track. This design is elaborated in Figure 4. The main challenge in designing such an RBL system is to ensure that correct system configurations cannot jump to invalid configurations. For example, if W did not maintain a foothold on both α and β during an $\alpha \to \beta$ step, then it would be possible to "jump" into either of the other step processes. To assist in this, a computational suite (to be released in the future) for designing and testing RBL schemes was developed. To finish, we prove (Theorem 1, Appendix A) that the designed scheme has the desired properties:

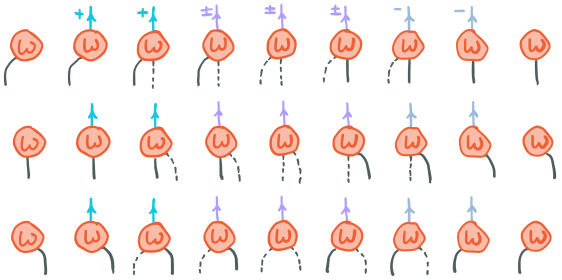
▶ **Theorem 1.** *The dynamics of the biased walker, in the long-run, are that of a biased random walk.*

4 Data and Computation

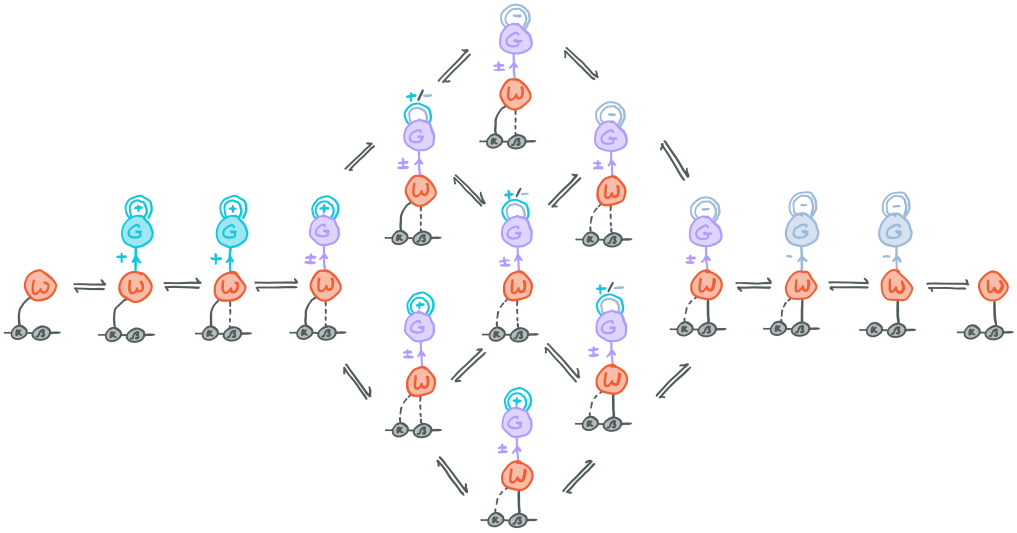
The advantages of structured data and programming abstractions are well known to users of high-level programming languages. In this section, we will develop a set of conventions and motifs for representing and computing with structured data in RBL. We will use these to implement three example "programs": (1) logical negation of a Boolean value, with which we will introduce **conditional branching**; (2) addition of natural numbers (using a Peano representation), with which we will introduce **looping**; (3) squaring of natural numbers, using addition as a **subroutine**. A recursive implementation of addition is left as an exercise for the reader. High level schemata to motivate these implementations are presented in Figure 5.

(a) The track.

(b) The walker, W.



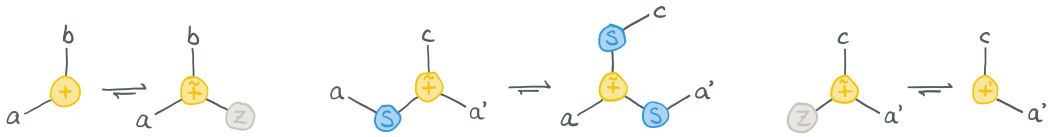
(c) The permissible energy configurations of W ($E = 0$).



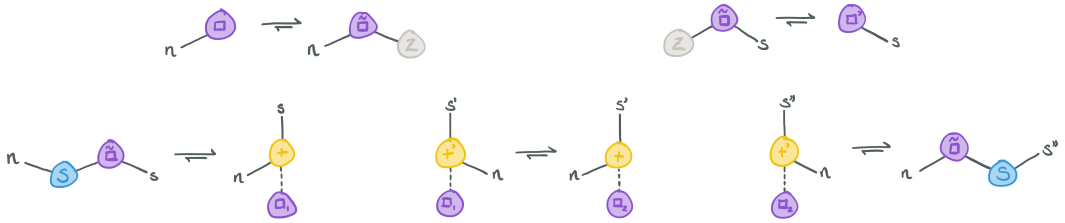
(d) The transition diagram for an $\alpha \to \beta$ step, i.e. $\mathsf{W}_\alpha + \mathsf{G}_+ \rightleftharpoons \mathsf{W}_\beta + \mathsf{G}_-$.

Figure 4 An RBL implementation of a biased walker. **(a)** The track consists of a polymer of α, β, and γ track monomers. Their implementation is nearly identical to that of the track in Figure 2a, except that: (1) the L/R ports enforce the $\cdots\alpha\beta\gamma\alpha\cdots$ sequence, by R_α/L_β, R_β/L_γ, and R_γ/L_α each having a distinct port type; and (2) each monomer's foothold port has a distinct type (α, β, or γ). Another difference is that the foothold ports have two bond colors: "solid" and "dashed", with the latter corresponding to a transitional state. The track can again be linear or cyclic. **(b)** The walker atom W has three feet which specifically bind each type of track monomer. It also has a fuel coupling port f. **(c)** The set of permissible energy configurations of W implements each of the "reactions" in Equation (1), one per row. Considering the top row, we start with W bound to an α track monomer. If it then binds to G_+, it can begin to attempt an $\alpha \to \beta$ step (looking to the end of the third row, if it instead bound G_- then it would begin to attempt an $\alpha \to \gamma$ backstep). From here, it tentatively (dashed bond color) places a foot on the β monomer. Then it changes the color of f to $\pm$ (lilac), the transitional state. While the fuel is interconverting between G_+ and G_-, it changes its α binding to dashed and its β binding to solid in two steps. Then, it attempts to change the color of f to $-$ (gray-blue), which would indicate the successful consumption of fuel ($\mathsf{G}_+ \to \mathsf{G}_-$). It can then dissociate its α foot followed by the spent fuel G_-. While we have described this step as if W has "intent", it is important to remember that this is only for narrative benefit; the system is really performing a random walk through configuration space, with bias provided by the free energy stored in the concentration of the fuel. **(d)** This $\alpha \to \beta$ step process is expounded in the transition diagram, which shows which system configurations can be reached from which other system configurations. This makes clear that the consumption of fuel and transition of foothold states occur in parallel.

(a) Logical negation. A Boolean value, T or F, is tagged with an atom (¬) which serves as a "reference" to the logical negation "function". Then, two "reactions" are implemented recognizing each of the possible inputs. The return value is tagged with ($\neg'$) to represent the result of the logical negation.

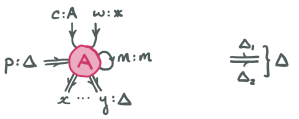


(b) Addition. As with logical negation, we use two special atoms to represent an addition to be performed, (+), and the result of an addition, ($+'$). As the computation must be reversible, we retain one of the inputs: for example, after adding $a = 3$ and $b = 4$, we would return $c = 7$ and $a' = 3$. The inputs and outputs are represented as Peano numbers, i.e. polymers of the form $\mathsf{S}-\mathsf{S}-\cdots-\mathsf{Z}$ where the number of S atoms corresponds to the number. For example, $3 \equiv \mathsf{S}-\mathsf{S}-\mathsf{S}-\mathsf{Z}$. The core of the computation is a loop (middle reaction, tagged by ($\tilde{+}$)), which iteratively decrements the bottom-left number and increments the top and bottom-right numbers. To see that this reaction functions as a loop, notice that the product can serve as a reactant to the same reaction while the loop condition holds. Note also how any possible molecule can participate in at most two reactions (corresponding to a forward and backward transition), giving a deterministic path from input to output. As the bottom-left number is originally a and the top number is originally b, these become respectively 0 and $c = a + b$. The bottom-right number starts at 0 and becomes $a' = a$. The left reaction is responsible for entering the loop, and the right reaction for exiting the loop.

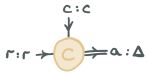


(c) Squaring. A number n can be squared by using the identity $n^2 = (2n-1) + (2n-3) + \cdots + 3 + 1$: that is, n^2 is the sum of the first n odd numbers. The implementation of squaring uses (□) and ($\square'$) as initial and final tags. It takes a Peano number n as input and returns a Peano number $s = n^2$ as output. We implement squaring using a loop (tagged by ($\tilde{\square}$)) with two variables, n and s. Initially, n is the number to be squared and $s = 0$. On each iteration, we decrement n, add this to s twice, and then increment s; the net result is $s \mapsto s + (2n-1)$ and $n \mapsto n - 1$. The addition is performed by using the program in Figure 5b as a subroutine. By starting with the largest odd number and ending at 1, we ensure that we consume the value n (by reducing it to 0) and produce only $s = n^2$ as an output, even though addition always returns one of its inputs.

Figure 5 High level schematic representations of three simple programs. These are not themselves RBL systems, but will be used as a blueprint for the construction of equivalent RBL systems. Biased arrows indicate the preferred direction for forward computation.



(a) The general form of a data-atom and data-port (Δ). Its c and w ports correspond to the specific and wildcard control ports respectively, and the m port pair corresponds to the "monomer" self-loop. The p data-port binds the data-atoms parent, and $x \cdots y$ provide bindings to some number of children.



(b) The C atom. The a data-port binds some computational data, the r port binds its root or origin, and the c port may be bound by some compuzyme.

Figure 6 The RBL atoms for data-atoms and the C atom. The energy configurations and dynamics are explored in Appendix B.

4.1 Motifs

Recall that we intend for the computational systems we design to be modular, compositional, and to support component reuse. To realize these properties, we will require a common convention for the representation of data. Moreover, as this data may be reused in different contexts (for example, Peano numbers may be added or multiplied), the data itself should generally be inert. Computation will be performed by dedicated catalytic "machines", taking inspiration from biochemical systems. We will call these "compuzymes" (computational enzymes). Consider the schematic for addition (Figure 5b). In principle, we will be able to construct RBL molecules corresponding to each of the three types of abstract molecule in the scheme, and we can also construct a compuzyme implementing each of the three reactions. To demonstrate the power of RBL, however, we will construct a single compuzyme performing the entire addition (loop included).

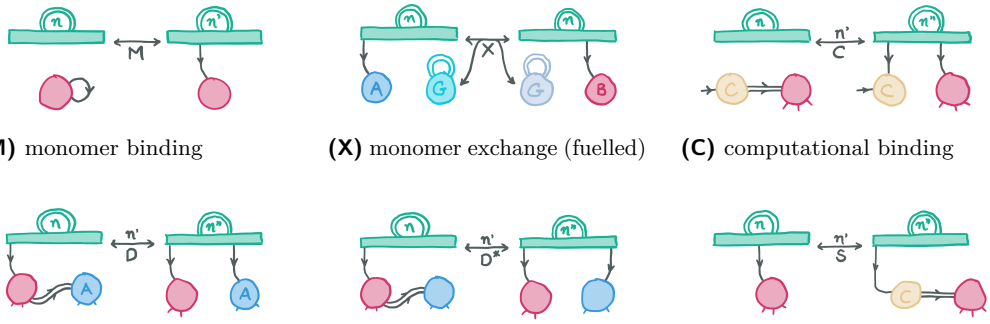
Data will be constructed from "data-atoms", RBL atoms with a certain structure. Consider Peano numbers, where a number is either zero (Z) or the successor (S) of another number. In haskell, this may be represented as `data` $\mathbb{N}$ `= Z | S` $\mathbb{N}$. We can therefore see that the S atom has one "child" while the terminal Z atom has no children. For an example of a data-atom with more than one child, consider the nodes of a binary tree[1]. These data-atoms would have three children, two for its child nodes (or leaves), and one for its associated data. As all data-atoms could be the child of another atom, each data-atom also has a parent. Therefore, an RBL representation of a data-atom must have an in-port for a parent to bind, and an out-port for each of its children; we refer to these special ports as "data-ports". These ports will in fact be pairs of ports: as with the double self-loop on the fuel atom G, a pair of bonds can be used to provide an energy barrier against unintentional bond-breakage.

In addition to parent and child data-ports, a data-atom needs a few other ports. To manipulate data, we add two control ports. These allow compuzymes to externally signal data-atoms to alter their configuration. One control port is specific to the type of data-atom; for example, the control port of an S atom is distinct from that of a Z atom. The other control port is a "wildcard" control port that is common to all data-atoms, and enables (limited) manipulation of variable data. Lastly, we have a pair of ports that can form a self-loop. This self-loop is present only on free monomers. The reason why this is required is somewhat subtle, and is explained in Appendix B.

The final key component of data is the C atom. Some data represent computations (either to be performed or that have completed), such as the molecules in Figure 5. These data are capped by a C atom, e.g. C–(¬)–T. The C atom therefore identifies computational data. It may seem extraneous, but it provides important indirection for subcomputations by allowing two compuzymes to interact with the same computational data simultaneously. As such, C has a "root" port (r) bound by the computation's origin, a "compuzyme" port (c) optionally bound by a compuzyme, and a data-port (a) binding the actual computational data.

The RBL atoms for data-atoms and the C atom are shown in Figure 6. These data structures can then be operated on by "compuzymes" to perform computation. Compuzymes are special atoms that behave as a platform for structural manipulation, and are typically represented as a rectangle. By recruiting data-atoms and interacting with them via their control ports, a sequence of structural manipulations can be effected. Compuzymes also maintain an internal state, via the color of a pair of self-loops. These state colors are typically numbered, but any label will suffice. Internal states are useful for distinguishing between

[1] Recall that the definition of a binary tree in haskell is `data Tr` a `= Lf | Nd (Tr` a`)` a `(Tr` a`)`.



(M) monomer binding **(X)** monomer exchange (fuelled) **(C)** computational binding

(D) destructuring **(D*)** wildcard destructuring **(S)** subcomputation initiation

Figure 7 The five core motifs underlying our computational convention. Only the net effect of the motif is shown; the detailed RBL implementations are given in Appendix B. The green rectangle is a compuzyme, and the encircled value n corresponds to the internal compuzyme state. In all motifs except X, there are one or two compuzyme state changes (from n to n', and possibly to n''). **(M)** The binding of a free monomer (pink data-atom). **(X)** The exchange of a monomeric data-atom A for B. **(C)** The binding of some computational data (pink data-atom, possibly with children). **(D,D*)** The destructuring of a child (blue data-atom) from its parent (pink data-atom). In the first case, the identity of the child data-atom (A) is known by the compuzyme, while in the second case it is not. **(S)** The initiation of a subcomputation. The pink data-atom (and children) represents some pre-prepared computational data. It is bound to a fresh C atom, exposing it to the action of other compuzymes.

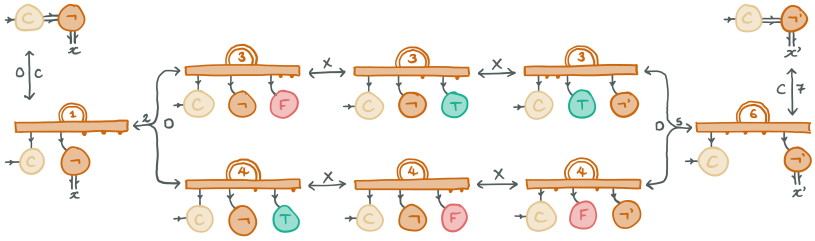
similar configurations, such as may occur during branches, loops, or long sequences. There are five core compuzyme motifs that are needed to perform arbitrary manipulations. These are shown in Figure 7, and their detailed implementations may be found in Appendix B, along with a description of the bond colorings of the data-ports and control ports. As RBL is reversible, each of these motifs may also act in reverse; for example, Motif D may also be used to bind a child data-atom to a parent. By convention, Motif X also expends one unit of fuel to drive computation forward, but in principle fuel coupling can occur at any point(s) during compuzyme operation.

To construct a compuzyme in RBL, we should first prepare an (abbreviated) transition diagram of its operation using the above motifs. Having done this, we will be able to determine what ports the compuzyme will require. With this, a suitable RBL atom can be designed. Then, the RBL implementations of each motif (shown in Appendix B) dictate which configurations are possible. For the most part, each motif will correspond to a distinct compuzyme state, and so the sets of possible configurations will generally be disjoint. We need merely extend the configurations to include the state of the other compuzyme bonds, which will be static within the motif. In some cases, particularly with branching control flow, motifs may share configurations. Thus the complete RBL description of the compuzyme is simply the union of all the configurations of the motifs.

4.2 Logical Negation

With these motifs we are now in a position to implement our three example programs. One approach to implement logical negation is to prepare two compuzymes performing each of the two "reactions" in Figure 5a, $\mathsf{Not_T}$ and $\mathsf{Not_F}$ respectively. The compuzyme implementing $\neg\mathsf{T}$ would then use Motif C to bind a $(\neg)$ computational data-atom, and then Motif D specialized to T. It can then use Motif X twice to swap T for F and $(\neg)$ for $(\neg')$, followed

(a) The RBL atoms for logical negation. From left to right, the compuzyme Not, the computational data-atoms ($\neg$) and ($\neg'$), and the data-atoms for Boolean values, F and T. Not has ports for fuel (f), atom C (C), and the control ports of each of the data-atom types; port labels and types are the same.



(b) The abbreviated transition diagram for logical negation, showing the motifs employed. The compuzyme Not starts and ends in its pure unbonded atomic state, and so acts catalytically. The f port is not shown, and unbound ports are shown with a dot. To aid comprehension, port order is consistent with that shown in (a).

Figure 8 The implementation of logical negation using conditional branching.

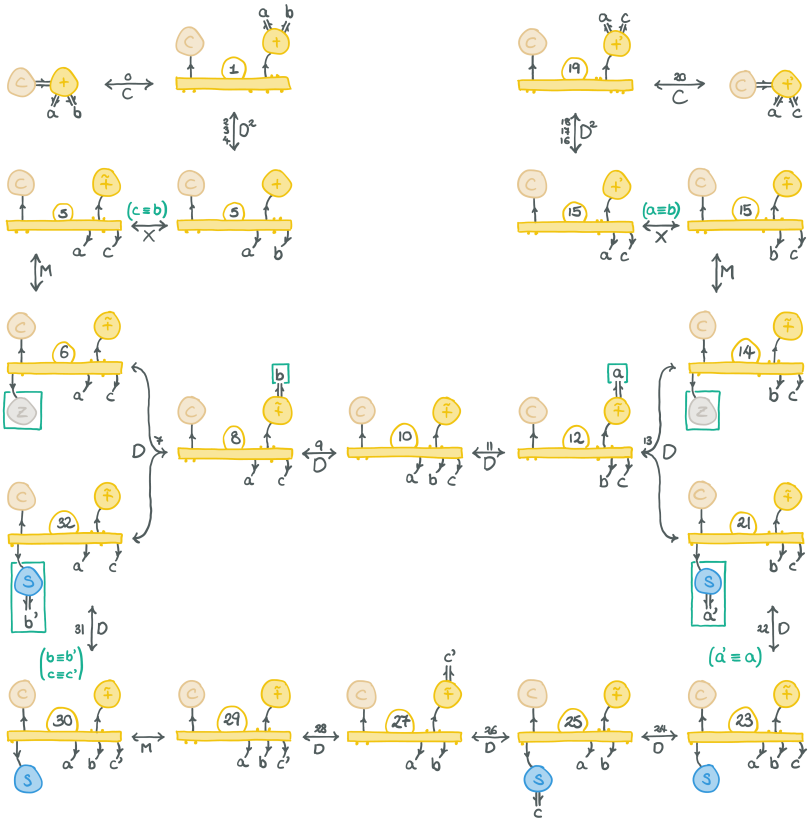
by Motif D to bind F to ($\neg'$), and finally Motif C to eject the result of the computation. The compuzyme for $\neg$F would be very similar. Of course, it is possible that compuzyme Not_T might inadvertently bind C–($\neg$)–F, but the specificity of Motif D would mean the compuzyme stalls. The only available option would be for it to backtrack; only the second compuzyme can complete the computation in this case. Moreover, only once the correct input to the compuzyme is bound can it expend fuel.

However, RBL allows us to go beyond this and implement conditional branches in control flow without backtracking. That is, we can create a single compuzyme Not implementing both cases. We can overlap two (or more) instances of Motif D, each recognizing different data-atoms. The motifs would use the same states n and n', but differing states n'' for each case; these distinct states n'' then allow us to distinguish the different branches of control flow. This is shown in Figure 8. Indeed, the dynamics of Not are simply the conjunction of those for Not_T and Not_F. Note that, when the two branches of control flow are later merged, this is a reversible operation: the two branches have to be *clearly* distinct. Here this is the case because we are using Motif D against two cases, T and F, "combining" the data-atoms into a single variable data-atom x'.

4.3 Addition

Similarly, we could implement addition using three compuzymes as indicated in Figure 5b. However, we again leverage the power of RBL to show how looping can be implemented directly with a single compuzyme Add. This implementation is given in Figure 9. Note that in this condensed implementation, the atom ($\tilde{+}$) now just serves to hold temporary variables and so only has one child compared to three in our original schematic. Looping is not much different from the control flow for logical negation; whereas for conditional branching, we had a branch followed by a merge, here we have a merge followed by a branch. At the beginning of a loop, we merge control flow from two branches: entry and loop-continuation. This would usually be trivial, but in a reversible context we need to be careful to conditionally distinguish these two events (so that, in reverse, we know when to "un-start" the loop). At the end of a loop, we branch into exit and loop-continuation.

(a) The RBL atoms for addition. From left to right, the compuzyme Add, the computational data-atoms $(+)$, $(\tilde{+})$, and $(+')$, and the data-atoms for Peano numbers, S and Z. Add has ports for fuel (f, not shown), atom C, and control ports for each type of data-atom, but also wildcard control ports a, b, and c for holding variable values during computation; port labels and types are the same, except for the variables which are of type $*$.



(b) The abbreviated transition diagram for addition, showing the motifs employed. Sometimes, two Motif Ds are elided into a single transition (shown as D^2). Green annotations show where variables are renamed between transitions or where two alternative data-atoms are bound to a variable position. To aid comprehension, port order is consistent with that shown in (a) except that atoms above the compuzyme are flipped vertically; additionally, unbound ports are shown with a dot, and the compuzyme state is shown with a single circle. Addition is implemented as a loop where the value of a is added to both b (renamed to c, and yielding $c = a + b$) and 0 (yielding a copy of a). The top left set of transitions enter the loop. The center set of transitions implement the control flow of the loop: on the left, they merge branches corresponding to loop entry and loop continuation, using b to discriminate these branches; on the right, they branch into the exit path or the main loop body, using a to discriminate these branches; the middle transitions tidy up/set up these branch operations. The bottom set of transitions implement the loop body, decrementing a and incrementing both b (now representing the copy of a) and c.

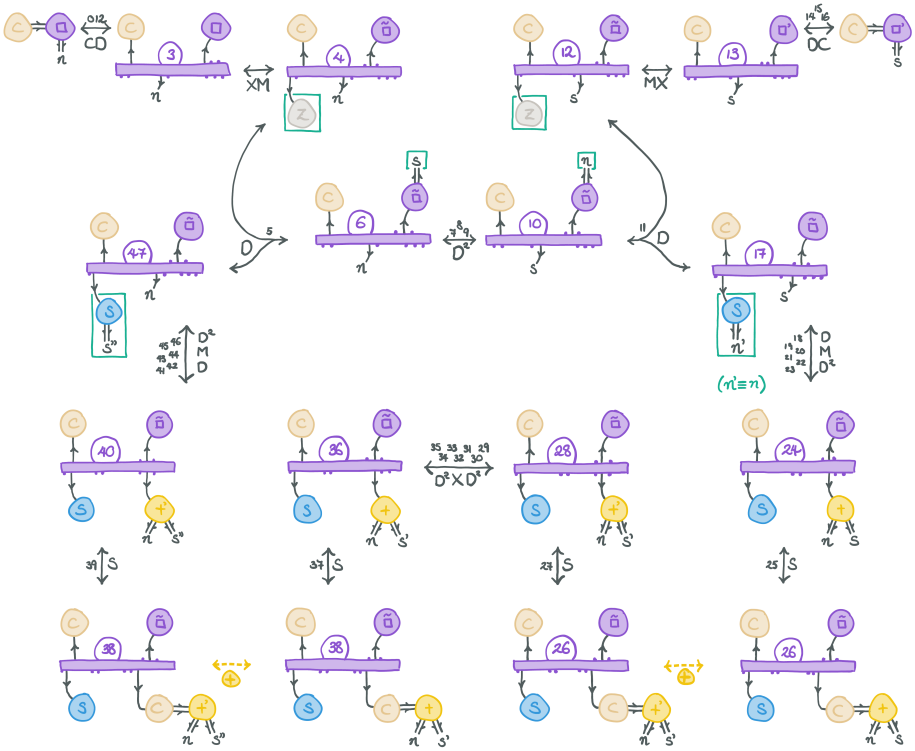
Figure 9 The implementation of addition using looping.

In our implementation of addition, we start with (a, b) and end with (a, c) where $c = a + b$. We begin by renaming b to c, to which we will add the value of a. Then we set the now free variable b to 0, to which we will also add the value of a. In our loop, the entry condition is therefore that the accumulator (b) is 0; if it is greater, then we have performed at least one iteration. The exit condition is that $a = 0$, as we decrement it on each iteration. Therefore, we merely need to use Motif D to check whether these variables are Z or S – n.

Due to reversibility, this program can also perform subtractions. To achieve this, one could either reverse the bias of the free energy currency, duplicate the implementation with reversed fuel coupling, or introduce a "switch" for the direction of computation.

4.4 Squaring

(a) The RBL atoms for squaring. From left to right, the compuzyme Sq and the computational data-atoms (□), (□̃), and (□′). Sq has ports for fuel (f, not shown), atom C, and control ports for the computational data-atoms shown, Peano data-atoms, two variables (n and s), as well as a set of ports for the addition subroutine: the computational data-atoms (+) and (+′) and the atom C; port labels and types are the same, except for the variables which are of type $*$.



(b) The abbreviated transition diagram for squaring, showing the motifs employed. Some motifs are elided into a single transition for brevity. Green annotations show where variables are renamed between transitions or where two alternative data-atoms are bound to a variable position. To aid comprehension, port order is consistent with that shown in (a) except that atoms above the compuzyme are flipped vertically; additionally, unbound ports are shown with a dot, and the compuzyme state is shown with a single circle. The implementation of squaring consists of a loop which on each iteration maps $s \mapsto s + (2n - 1)$ and $n \mapsto n - 1$, with s initially 0 and n ending at 0. The loop structure is essentially the same as for addition (Figure 9). To perform the map $s \mapsto s + (2n - 1)$, we first decrement n, then add the new value of n to s twice before incrementing it. The additions are performed using Add by preparing computational data representing the two additions.

Figure 10 The implementation of squaring using subroutines.

Finally, we implement squaring. Again, squaring consists of a loop; however, this loop involves calling addition as a subroutine. This is quite simply done. We use the data manipulation motifs to bind the numbers we wish to add to a (+) atom; then Motif S binds this computational data to a C atom, thus "presenting" it to Add compuzymes. The implementation is shown in Figure 10.

As with addition, the reverse of this program is also useful and performs square roots (over a suitably restricted domain – attempting to take the square root of 10 would lead to something resembling a runtime error).

4.5 Reversible Turing Machines

The implementation of reversible Turing Machines (defined by Bennett [2]) is left as an exercise for the reader. It only requires conditional branching, but the reversible implementation of a bi-infinite tape requires some care. For a hint, a reference implementation in a related language is given in the author's thesis [9] (p. 147).

5 Discussion

As required, the RBL model admits a variety of forms of component reuse: monomers used in the construction of molecules can be fully recycled, being drawn from and returned to a pool of free components; dynamic components can be designed to act catalytically, and so may be reused multiple times; and the same monomers (e.g. those representing Peano numbers) can be reused in distinct contexts without interference. These properties are strengthened by the modularity conferred by molecular structure: any number of instances of the same program or system can run in parallel using the same components and without crosstalk; and common sub-systems can be factored out and used by multiple distinct systems, such as a single common fuel species supplying all the free energy in the system. Moreover, RBL is particularly amenable to implementing hierarchies of abstraction. For instance, we saw the following hierarchy of computational abstractions: the definition of RBL "data-atoms"; a set of basic structural-manipulation motifs; control flow primitives such as branching and looping; and subroutines such as addition, which could then be used as a black-box by the squaring routine. Further abstractions which are possible but not shown include recursion and concurrency.

While RBL is a powerful model, it also has some limitations in its current formulation. Simulations of the squaring routine encounter 1700 distinct configurations when computing 3^2 (alternatively, $\sqrt{9}$). This is a reflection of the hierarchy of abstractions used; the structural manipulation motifs incur a cost of $\sim$ 10–15 transitions each, and multiple motifs must be employed to perform a given structural manipulation leading to an overhead on the order of 50–100 transitions for each useful computational step. Simpler "machines" such as the biased walker lie closer to "pure" RBL and incur 8–10 transitions per step (counting the shortest path in Figure 4d, and depending on whether we count the parallel transitions separately), though this is still perhaps larger than desired. Arguably the most significant reason for these overheads originates in the lack of spatial/geometric awareness of RBL. In biochemical systems, enzymes are able to take advantage of shape complementarity and differently-shaped conformations to perform their manipulations. This description omits the series of microstate transitions involved in an enzymatic reaction, but even so enzymes are (usually) particularly fast and efficient. Meanwhile in RBL we must very carefully coordinate structural manipulations through the use of, e.g., control ports: our lack of spatial awareness means that, without such coordination, we could not selectively displace a particular child data-atom from its parent. This suggests an obvious RBL variant to pursue in the future: one in which geometry plays a first-class role. Such a model should be able to more directly perform desired structural manipulations. Moreover, incorporating geometry would ensure that our designs are physically possible: the non-geometric RBL model can easily encode structures that are too crowded for Euclidean space, such as a complete binary tree of depth 20.

Another challenge for the experimental realization of RBL is that the number of configurations of an RBL atoms scales exponentially with the number of ports, and each of these may have a distinct energy. Restricting the set of allowed energies E to $\{0, \infty\}$ would considerably simplify this, as we may be able to get away with just programming the allowed configurations. Nevertheless, compuzymes such as Add and Sq have on the order of 200 allowed configurations. Furthermore, control ports and compuzyme states involve dozens of possible bond colors. Future work should investigate whether the number of required bond colors can be reduced, or whether bond colors are even required (i.e. whether ports can be monochromatic). Another question is whether atom configurations can be "factored" into simpler subsystems. Additionally, it would be useful to know what the minimum required complexity is to implement useful computational abstractions. For example, while Add may require ~ 200 configurations, the division into three compuzymes $\mathsf{Add}_{\text{init.}}$, $\mathsf{Add}_{\text{loop}}$, and $\mathsf{Add}_{\text{fin.}}$ may be substantially simpler. Combining with a geometric variant of RBL should also reduce the number of distinct configurations required.

Lastly, a more accurate treatment of the kinetics of RBL is warranted. Not only will configuration energy contribute to transition rates, but also any associated entropy changes. These entropy changes are particularly significant when particle number changes, such as when a compuzyme binds some computational data. There are also entropy changes associated with interactions with the monomer pools that cannot be eliminated through any "clever" means (see the author's thesis [9] (pp. 101–120)).

There is some similarity between RBL and Thermodynamic Binding Networks (TBNs) [8]. TBNs consist of monomers with a geometry-free collection of domains, similar to RBL atoms with a geometry-free collection of ports. Domains have complementary codomains, and these can form bonds. However, the TBN model explicitly does not consider the kinetics of system evolution. Instead it focuses on analyzing the possible stable configurations admitted by a given TBN to determine whether leak reactions are possible, and presumes that the desired state corresponds to thermodynamic equilibrium. In this way, the goals of TBNs diverge from the goal of RBL. In RBL, it is the programming of kinetic pathways that is important, and equilibration is to be avoided. Nevertheless, a realistic implementation of RBL would likely be subject to error conditions such as leak reactions, and so it would be interesting to use a TBN-like framework to evaluate or improve the robustness of RBL systems. Other related systems worth comparing in the future include Polymer Reaction Networks [15] and graph-theoretic molecular systems [18, 31].

In conclusion, RBL is an interesting new model with powerful properties, but it is also currently too unwieldy for experimental realization. Future work will further develop RBL, including variant models, to determine the possibility of achieving these powerful properties in a real chemical system.

References

1 Leonard M Adleman. Molecular computation of solutions to combinatorial problems. *science*, 266(5187):1021–1024, 1994.

2 Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532, 1973.

3 Charles H Bennett. The thermodynamics of computation—a review. *International Journal of Theoretical Physics*, 21:905–940, 1982.

4 Tatiana Brailovskaya, Gokul Gowri, Sean Yu, and Erik Winfree. Reversible computation using swap reactions on a surface. In *DNA Computing and Molecular Programming: 25th International Conference, DNA 25, Seattle, WA, USA, August 5–9, 2019, Proceedings 25*, pages 174–196. Springer, 2019.

5 Rory A Brittain, Nick S Jones, and Thomas E Ouldridge. What would it take to build a thermodynamically reversible universal turing machine? computational and thermodynamic constraints in a molecular design. *arXiv preprint arXiv:2102.03388*, 2021.

6 Anne Condon, Alan J Hu, Ján Maňuch, and Chris Thachuk. Less haste, less waste: on recycling and its limits in strand displacement systems. *Interface Focus*, 2(4):512–521, 2012.

7 Erica Del Grosso, Elisa Franco, Leonard J Prins, and Francesco Ricci. Dissipative DNA nanotechnology. *Nature Chemistry*, 14(6):600–613, 2022.

8 David Doty, Trent A Rogers, David Soloveichik, Chris Thachuk, and Damien Woods. Thermodynamic binding networks. In *DNA Computing and Molecular Programming: 23rd International Conference, DNA 23, Austin, TX, USA, September 24–28, 2017, Proceedings 23*, pages 249–266. Springer, 2017.

9 Hannah A Earley. *On the performance and programming of reversible molecular computers*. PhD thesis, University of Cambridge, 2021.

10 Abeer Eshra, Shalin Shah, Tianqi Song, and John Reif. Renewable DNA hairpin-based logic circuits. *IEEE Transactions on Nanotechnology*, 18:252–259, 2019.

11 Constantine G Evans and Erik Winfree. Physical principles for DNA tile self-assembly. *Chemical Society Reviews*, 46(12):3808–3829, 2017.

12 Anthony J Genot, Jonathan Bath, and Andrew J Turberfield. Reversible logic circuits made of DNA. *Journal of the American Chemical Society*, 133(50):20080–20083, 2011.

13 SJ Green, Jonathan Bath, and AJ Turberfield. Coordinated chemomechanical cycles: a mechanism for autonomous molecular motion. *Physical review letters*, 101(23):238101, 2008.

14 Hope A Johnson and Lulu Qian. Simplifying chemical reaction network implementations with two-stranded DNA building blocks. In *26th International Conference on DNA Computing and Molecular Programming (DNA 26)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

15 Hope A Johnson and Erik Winfree. Verifying polymer reaction networks using bisimulation. *Theoretical Computer Science*, 843:84–114, 2020.

16 Hope Amber Johnson and Anne Condon. A coupled reconfiguration mechanism for single-stranded DNA strand displacement systems. In *28th International Conference on DNA Computing and Molecular Programming (DNA 28)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

17 Jocelyn Y Kishi, Thomas E Schaus, Nikhil Gopalkrishnan, Feng Xuan, and Peng Yin. Programmable autonomous synthesis of single-stranded DNA. *Nature chemistry*, 10(2):155–164, 2018.

18 Eric Klavins. Directed self-assembly using graph grammars. *Foundations of nanoscience: self assembled architectures and devices, Snowbird, UT*, 2004.

19 Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM journal of research and development*, 5(3):183–191, 1961.

20 Christopher Lutz and Howard Derby. Janus: a time-reversible language. *Letter to R. Landauer*, 2, 1986.

21 Kevin Montagne, Raphael Plasson, Yasuyuki Sakai, Teruo Fujii, and Yannick Rondelez. Programming an in vitro DNA oscillator using a molecular networking strategy. *Molecular systems biology*, 7(1):466, 2011.

22 Jennifer E Padilla, Matthew J Patitz, Robert T Schweller, Nadrian C Seeman, Scott M Summers, and Xingsi Zhong. Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. *International Journal of Foundations of Computer Science*, 25(04):459–488, 2014.

23 Lulu Qian, David Soloveichik, and Erik Winfree. Efficient turing-universal computation with DNA polymers. In *DNA Computing and Molecular Programming: 16th International Conference, DNA 16, Hong Kong, China, June 14-17, 2010, Revised Selected Papers 16*, pages 123–140. Springer, 2011.

24 Ian Seet, Thomas E Ouldridge, and Jonathan PK Doye. Simulation of reversible molecular mechanical logic gates and circuits. *Physical Review E*, 107(2):024134, 2023.

25 Jong-Shik Shin and Niles A Pierce. A synthetic DNA walker for molecular transport. *Journal of the American Chemical Society*, 126(35):10834–10835, 2004.

26 Friedrich C Simmel, Bernard Yurke, and Hari R Singh. Principles and applications of nucleic acid strand displacement reactions. *Chemical reviews*, 119(10):6326–6369, 2019.

27 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *natural computing*, 7:615–633, 2008.

28 David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

29 Leo Szilard. Über die Entropieverminderung in einem thermodynamischen System bei Eingriffen intelligenter Wesen. *Zeitschrift für Physik*, 53(11-12):840–856, 1929.

30 Anupama J Thubagere, Wei Li, Hope A Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, et al. A cargo-sorting DNA robot. *Science*, 357(6356):eaan6558, 2017.

31 Kohji Tomita, Haruhisa Kurokawa, and Satoshi Murata. Graph-rewriting automata as a natural extension of cellular automata. *Adaptive Networks: Theory, Models and Applications*, pages 291–309, 2009.

32 Erik Winfree. *Algorithmic self-assembly of DNA*. California Institute of Technology, 1998.

33 Peng Yin, Harry MT Choi, Colby R Calvert, and Niles A Pierce. Programming biomolecular self-assembly pathways. *Nature*, 451(7176):318–322, 2008.

34 Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Principles of a reversible programming language. In *Proceedings of the 5th Conference on Computing Frontiers*, pages 43–54, 2008.

35 David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, 2011.

A Proofs

▶ **Theorem 1.** *The dynamics of the biased walker, in the long-run, are that of a biased random walk.*

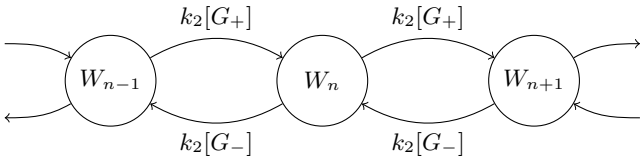
Proof. Recall the definition of the biased walker (Section 3.3) and its transition diagram (Figure 4d). We will assume typical CRN dynamics, namely that the evolution of the system may be described by a Continuous-Time Markov Chain (CTMC) with transition rates given by the law of mass action. For example, the reaction $W_\alpha + G_+ \rightleftharpoons W_\alpha{:}G_+$ has forward transition rate $k_2[W_\alpha][G_+]$ and reverse transition rate $k_1[W_\alpha{:}G_+]$ where $[X]$ is the concentration of species X, k_1 is the rate constant for unimolecular reactions, and k_2 that for bimolecular reactions (all the species have equal energy, and we assume no activation energy). Note the periodic symmetry of the walker scheme; without loss of generality, we can relabel the transition diagram to go from W_n to W_{n+1} where n corresponds to the n^{th} position along the track. We assume the track extends infinitely in both directions, so that $n \in \mathbb{Z}$. We first prove that the net reaction $W_n \rightleftharpoons W_{n+1}$ has rate constants $\propto [G_+]$ and $\propto [G_-]$ for the forward and reverse reactions.

The dynamics of the walker are somewhat complicated by the number of intermediate states – 13 between each net step (there are 15 distinct states in Figure 4d: 13 intermediate states, and W_α and W_β). However, after some convergence time, any initial distribution will converge to a steady state distribution (up to periodicity of the Markov Chain (MC)); this is analogous to the steady state approximation in chemistry. To see this, note that the behavior within each step-window (between W_n and W_{n+1}) is identical. Therefore, we can overlap each of these "sub-chains", reducing the infinite MC to the finite MC from W_n to W_{n+1} by symmetry. As this MC is aperiodic, irreducible, and reversible, it admits a steady

state given by detailed balance and this steady state is reached exponentially fast. After this "burn-in" time, the initial distribution is effectively forgotten.

For any pair of adjacent intermediate states i and j (states of the form $W{:}G$), the transition rate for $i \to j$ is $k_1[i]$. Consequently, by detailed balance their steady state concentrations are all equal (within a given step-window), and we can write this concentration as $[W{:}G]$. The remaining reactions are $W_n + G_+ \rightleftharpoons W_n{:}G_+$ and $W_{n+1}{:}G_- \rightleftharpoons W_{n+1} + G_-$. Using the transition rates given earlier, $k_2[W_n][G_+] = k_1[W{:}G] = k_2[W_{n+1}][G_-]$, and hence $[W_{n+1}]/[W_n] = [G_+]/[G_-]$. Furthermore, the net forward reaction rate constant for $W_n \rightleftharpoons W_{n+1}$ is $k_2[G_+]$, and the reverse reaction rate constant is $k_2[G_-]$.

Having proved this, we can reduce the original CTMC to the effective CTMC



which is prototypical of a random walk with bias $b = ([G_+] - [G_-])/([G_+] + [G_-])$. The expected value of n at time t, given $n(0) = 0$, will be $k_2 bt$. The variance is non-trivial for a continuous-time process, but will be approximately $\sqrt{k_2 t[G_+][G_-]}/([G_+] + [G_-])$. ◀

B Data & Compuzyme Motifs

Missing from the description of data-atoms and data-ports in Section 4.1 are the bond colors and atom configurations.

Data-ports consist of two ports. The first port (counting clockwise) has two bond colors, solid and dashed/‡. The second port is monochromatic. This design allows for the controlled and coordinated breaking of bonds.

Control ports have more colors to allow for the intricate signaling required by the motifs. These colors are: m, for "monomer"; solid/neutral, a bond that has no signaling intent; dashed/‡, for transitional states; C•, C‡, and C○, for displacing a data-atom from its C parent; and x•, x‡, and x○ for each child data-port x, for displacing child data-atoms. •, ‡, and ○ indicate that the relevant entity is bound, is transitioning between bound and unbound, or is unbound, respectively.

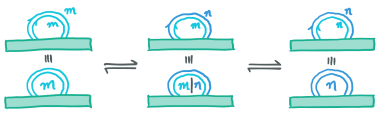
Wildcard control ports are much simpler, having just two bond colors: solid and dashed/‡.

With the bond colors enumerated, we now define the allowed RBL configuration for data-atoms and compuzymes. These are illustrated piecemeal in the remainder of this Appendix.

State changes

Not a motif in itself, compuzyme state changes are common to almost all our motifs. Compuzymes may perform a long series of manipulations, including branched control flow. As a result, indistinguishable configurations may be encountered, unexpectedly linking distinct parts of the computation. To prevent this, we imbue compuzymes with an internal "state" to track progress through configuration-space and correctly handle control flow. The

implementation is below:

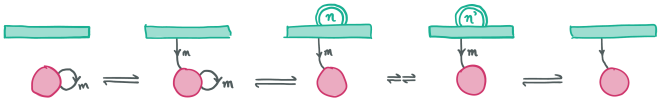


The top row is an RBL schematic, whereas the bottom row presents an abbreviated notation. The state is represented by a pair of self-loops. Each of these have the same set of bond colors, with each color representing a distinct state (e.g. m, n). Note that m and n are variables rather than explicit colors, i.e. m is not the same as the monomer color. Typically these states will be numbered, but any set of useful labels may be employed. There is also a special "$\varnothing$" state in which the bonds are broken; compuzymes typically start and end in this state. Changing state from m to n is simply a matter of changing the bond colors of the loops in turn.

The requirement of a pair of loops may seem superfluous; however, consider the case of two otherwise identical configurations that are meant to be distinguished by this state. If a single loop was used, then these configurations would be adjacent and so the state would not serve as a barrier. The double loop prevents this scenario, as the intermediate $m|n$ state will be constructed so as to be inaccessible.

M Monomer binding

Data molecules are formed from data-atoms, and so the manipulation of individual data-atom "monomers" is foundationally important. When building up data, fresh monomers will need to be drawn; conversely, when breaking down data the extraneous monomers will need to be discarded. We suppose that the environment provides an unlimited pool of fresh monomers for these purposes. Both drawing and discarding monomers can be implemented by a single motif, as they are inverses of each other and our system is reversible:



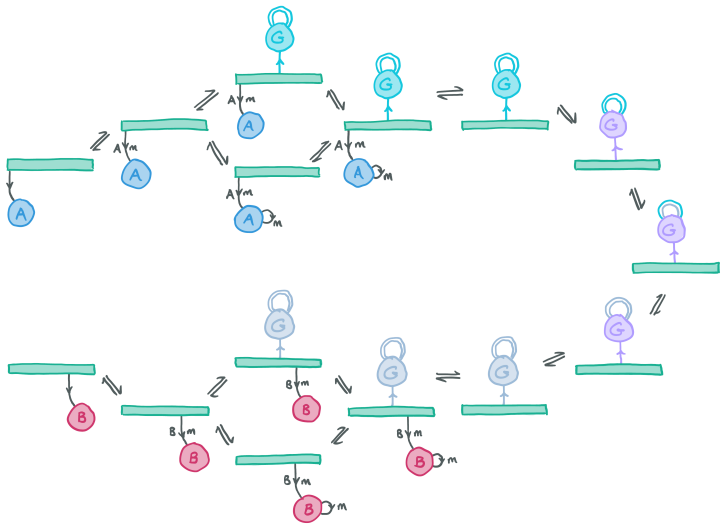
Here the monomer is the pink data-atom, and the compuzyme drawing (resp. discarding) the monomer is the green rectangle. In general we consider compuzymes as a platform for manipulations, hence the rectangular form. Both the free compuzyme and free monomer have an adjacent configuration in which the control ports form an "m-colored" bond. Consequently, the compuzyme is able to spontaneously form a bond with a free monomer of the correct type, drawing it from solution. Going forward, we will not explicitly mention adjacent configurations when they can be inferred from the diagrams. Recall that control ports are unique to a given data-atom type. As such, a compuzyme can be sure it is drawing the desired monomer from solution. We conclude the motif by switching the control port to the "neutral" bond color, ready for subsequent transitions.

Notice that the monomer in solution has a self-loop, also m-colored. Suppose that it didn't and instead had 0 bonds. In this case, the neutral bond in the final configuration could readily break. To address this, we assign an infinite energy to data-atoms with 0 bonds, and use the self-loop to mark monomers. Observe further that the compuzyme has no direct way to break the self-loop on the monomer. This is the reason for the existence of control ports: through different bond colors, we can signal different intents. Upon receiving such a signal, the normally-inert data-atom is able to transition to other configurations. Specifically,

there are usually $E = \infty$ energy barriers preventing a data-atom from transitioning to other configurations; but for specific control port colors, these energy barriers are lowered and a small region of configuration space is made available to explore. Note also the state change, the purpose of which we will elaborate further in Motif C.

X Monomer exchange

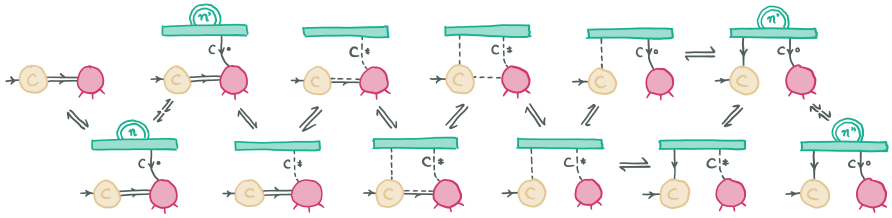
Related to the monomer binding motif is the monomer exchange motif. Arguably this is not a single motif in its own right, as it can be realized by combining two monomer-binding motifs back-to-back. However, monomer exchange is sufficiently common and useful that we promote it to its own motif. The implementation follows:



Notice that its implementation differs from two occurrences of Motif M: the state changes have been replaced by coupling to fuel. Though in principle any motif or state change can be coupled to the burning of fuel, we choose this motif as the canonical such point for convenience.

C Computational data binding

To bind computational data in order to manipulate it we need to (1) recognize it as such via its control port, and (2) displace it from its C parent:

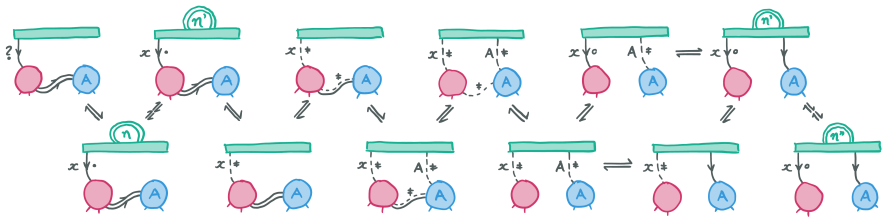


The C atom is in beige, and the computational atom in pink. Recall that the control port has three colors indicating progress of binding and displacement: C• corresponds to computational data with a C parent while C○ corresponds to computational data displaced from its parent. C‡ is a "transitional" state between these. We bind with C•, and then immediately change state from n to n'. These state changes are frequent in motifs and act as

bidirectional assertions. Without this, we could – for example – break the control port-bond during a transitional state. After this assertion, we switch to the transitional bond color C‡. In this configuration, the bond between computational atom and C parent can be weakened in a number of steps until it's broken completely. At the same time, we must form a bond with the soon-to-be-displaced C atom so as not to lose the association between them. This association is important, because this computation may be a subcomputation, with the particular C atom bound by another compuzyme. If we were to lose the association, then we could not return the result of this computation to the caller. Indeed, the series of adjacent configurations is programmed carefully so that the bond between C and the computational atom cannot be completely broken until the compuzyme has bound both. Finally we have another bidirectional assertion from state n' to n'', after which further transitions and motifs may occur. Notice that two of the last bond changes can happen in either order, hence the parallelism in the transition diagram.

D Destructuring

Data molecules are tree structures formed from data-atoms. To effectively manipulate these, we need to be able to break down (and build up) these structures into their constituent parts. We achieve this with the destructuring motif, which can be used to displace (resp. replace) a child data-atom from its parent. This motif is nearly identical to that of binding computational data, primarily differing in the control port bond colors used:



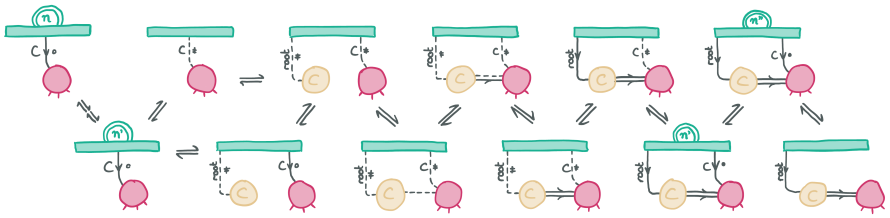
The ? bond "color" indicates that this motif doesn't care what the initial state of the control port is. Recall that a data-atom may have multiple children, each bound by a differently-labelled data-port. Here we are displacing a child data-atom (blue) from the x data-port of its parent (pink). Specifically, the child data-atom is of type A, and indeed this particular instance of Motif D can *only* displace an atom of type A. Note that the child data-atom may have its own children.

If we wish to introduce a branch (resp. merge) in control flow, then multiple of these motifs can be overlaid – one for each type of data-atom to be recognized. The first 5 configurations shown are common, as they are independent of the child data-atom type. The remaining configurations diverge. The nature of configuration adjacency ensures that this works as expected. We may also want to displace a data-atom regardless of its type. For this, we substitute the interactions with the control port for the wildcard control port.

S Subcomputation initiation

Because of the indirection provided by the C atom, compuzymes act independently of the context of a computation. This means that any routines we implement can also be used as subroutines. To call a subroutine, we prepare a "subcomputation": computational data that is bound to a host compuzyme and presented on a C atom. Other relevant compuzymes can then act on this subcomputation. Upon completion, the host compuzyme can accept the result and use it as required.

Prior to using this motif, the appropriate data representing the subcomputation should be prepared by means of the other motifs. Then, the subcomputation initiation motif will bind it to a fresh copy of the C atom:



This motif is specialized to the type of the data-atom. To complete the subroutine, we use another instance of this motif in reverse and specialized to the result data-atom. For example, if the initial data is tagged with $(+)$, then the final data would be tagged with $(+')$.

Now the purpose of the C atom becomes clear. We need to release the computational atom's control port so other compuzymes can interact with it, but then we would lose track of it. Therefore, we need to maintain a separate bond to this molecule. However, we cannot simply bind the data-atom on another port: it is important that this data-atom can be swapped out, for example $(+)$ for $(+')$, so as to indicate the completion of computation. As such, we use the intermediate atom C to achieve the necessary indirection to satisfy all of these conditions. In particular, we bind the C atom via its "root" port; its c port can then be bound by other compuzymes as needed.

Accelerating Self-Assembly of Crisscross Slat Systems

David Doty
University of California–Davis, CA, USA

Hunter Fleming
University of Arkansas, Fayetteville, AR, USA

Daniel Hader
University of Arkansas, Fayetteville, AR, USA

Matthew J. Patitz
University of Arkansas, Fayetteville, AR, USA

Lukas A. Vaughan
University of Arkansas, Fayetteville, AR, USA

Abstract

We present an abstract model of self-assembly of systems composed of "crisscross slats", which have been experimentally implemented as a single-stranded piece of DNA [21] or as a complete DNA origami structure [28]. We then introduce a more physically realistic "kinetic" model and show how important constants in the model were derived and tuned, and compare simulation-based results to experimental results [21, 28]. Using these models, we show how we can apply optimizations to designs of slat systems in order to lower the numbers of unique slat types required to build target structures. In general, we apply two types of techniques to achieve greatly reduced numbers of slat types. Similar to the experimental work implementing DNA origami-based slats, in our designs the slats oriented in horizontal and vertical directions are each restricted to their own plane and sets of them overlap each other in square regions which we refer to as *macrotiles*. Our first technique extends their previous work of reusing slat types within macrotiles and requires analyses of binding domain patterns to determine the potential for errors consisting of incorrect slat types attaching at undesired translations and reflections. The second technique leverages the power of algorithmic self-assembly to efficiently reuse entire macrotiles which self-assemble in patterns following designed algorithms that dictate the dimensions and patterns of growth.

Using these designs, we demonstrate that in kinetic simulations the systems with reduced numbers of slat types self-assemble more quickly than those with greater numbers. This provides evidence that such optimizations will also result in greater assembly speeds in experimental systems. Furthermore, the reduced numbers of slat types required have the potential to vastly reduce the cost and number of lab steps for crisscross assembly experiments.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases DNA origami, self-assembly, kinetic modeling, computational modeling

Digital Object Identifier 10.4230/LIPIcs.DNA.29.7

Funding *David Doty*: This author's work was supported by NSF grants 2211793, 1900931, and CAREER-1844976.
Hunter Fleming: This author's work was supported in part by NSF grant CAREER-1553166.
Daniel Hader: This author's work was supported in part by NSF grant CAREER-1553166.
Matthew J. Patitz: This author's work was supported in part by NSF grant CAREER-1553166.
Lukas A. Vaughan: This author's work was supported in part by NSF grant CAREER-1553166.

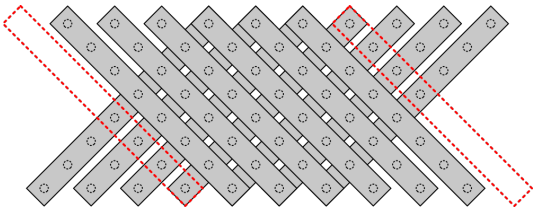


Figure 1 An illustration of a crisscross slat ribbon. Growth of the ribbon occurs in two layers with slats attaching to 4 binding domains presented by previously added slats in the opposite layer.

1 Introduction

In [21, 28], the authors introduced a novel scheme for the seeded self-assembly of DNA-based structures that is extremely resilient to spurious (unseeded) nucleation. In their scheme, structures are formed from individual components called *slats* that can be thought of as long, $1 \times n$ tiles with n binding domains distributed across their lengths. Unlike traditional DNA-based tiles that generally attach in a single plane to at most 2 adjacent tiles as they bind into an assembly, slats attach in multiple layers by the matching of binding domains between the layers. Consequently, whereas traditional tiles typically only coordinate their growth with at most two others, slats can achieve a much higher coordination by spanning across and binding with several slats in the other layer, as illustrated in Figure 1. This increase in coordination (a.k.a., cooperation) makes spontaneous growth away from seeded assemblies much more difficult, since this would require the entropically unfavorable merging of n slats simultaneously, an event with probability exponentially small in n.

While the work of [21, 28] was primarily interested in the robust nucleation properties of this crisscross scheme, here we explore crisscross slats as a means of growing structures that are robust to erroneous attachments while also using greatly reduced numbers of unique types of slats. In [28] they focused on systems of crisscross slats in which each slat is a complete DNA origami structure (depicted in Figure 2), and exhibited growth of structures that were of two types: (1) repeating patterns of slats forming unbounded 1D ribbons and 2D sheets (*periodic* structures), and (2) finite structures composed of unique slat types at each location (*hard-coded*, a.k.a., *fully addressable*, structures). They were able to build hard-coded structures that contained as many as 1,022 unique slat types. Although these structures had few errors (e.g., missing slats or slats in incorrect locations and/or translations or reflections), due to the low concentrations resulting from dilution after mixing so many different structures, the growth was very slow and even needed to be separated into distinct growth stages. In each stage, only a few hundred slat types were added and growth was allowed to continue for multiple days before the slat types for the next stage were added. Additionally, the authors estimated that without automated liquid handlers, manual pipetting of the many strand combinations to create the large numbers of slat types would require about one month of manual effort. Our techniques for lowering the numbers of slat types required to build target structures and patterns are intended to make growth of crisscross slat systems much faster (removing the need for staging) while experiencing similar error rates, and having additional benefits of making system designs cheaper and less labor intensive.

In this paper, we first introduce two models of crisscross slat-based self-assembling systems. These models are based on the abstract Tile Assembly Model (aTAM) and kinetic Tile Assembly Model (kTAM) introduced in [27]. The model based on the aTAM, which we call the *abstract Slat Assembly Model* (aSAM), is a mathematical abstraction suitable for creating and testing high-level designs, especially those of algorithmic self-assembling systems.

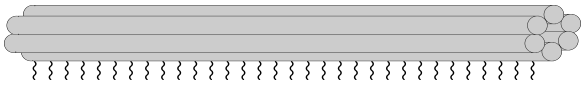


Figure 2 Schematic depiction of DNA origami slat as a 6-helix bundle. Each cylinder represents a DNA double helix, and the wavy lines underneath represent single-stranded "handles" (a.k.a., "glues") that serve as binding domains.

In algorithmic systems, the individual components (in this case, slats), can be thought of as simple program instructions. The binding of each slat effectively uses the domains available for a slat to bind (i.e. those exposed by slats already in the assembly and in the necessary geometric arrangement) to discriminate among slat types, allowing exactly one type to bind. Those initial binding domains can be thought of as the *input*, and the domains of the slat that remain exposed to which later slats can bind as the *outputs*. The design of a slat and its domains dictates the logic that transforms input into output and thus the function of the "instruction". In a computer program, instructions taken from a small number of unique instruction types can be executed many times each, processing information to determine which instructions are next and when the computation should end. In an algorithmic self-assembling system, a small set of slat types do the same thing. Much theoretical work has been done to show the power of algorithmic self-assembly [2, 8, 11, 14, 16–18, 20, 22, 27] and that structures can be built with optimally small sets of tile types [3–5, 7, 23, 26] .

The second model we introduce, the *kinetic Slat Assembly Model*, is based on the kTAM and intended to capture more physically realistic aspects of the self-assembly of systems. In this model, slat attachments are modeled as reversible processes, where the forward rates of attachment are dependent upon slat concentrations and the reverse (i.e. detachment) rates are dependent upon the number of bonds formed by a slat. In this way, the model is able to capture a wider spectrum of dynamics and model several types of errors that occur in experimental implementations.

We present two techniques for reducing the number of slat types used by systems. The first technique reduces number of unique slat types used within each (potentially repeating) square region referred to as a *macrotile*. In a slat system where the cooperativity value is n (i.e. each slat needs to bind with n others in order to join an assembly), we call each $n \times n$ region where n slats running horizontally overlap with n slats running vertically a macrotile. We present in Section 4.1 a technique for designing systems making use of multiple slats of the same type in each macrotile, thus reducing the overall slat type count, and then present the results from series of kSAM simulations demonstrating that such systems with fewer slat types can both assemble more quickly and can do so while maintaining low error rates.

Our second method of reducing slat type numbers is algorithmic self-assembly. To demonstrate this, we present our design for a system in which the slats compute the logical `xor` function, resulting in a system producing the discrete self-similar fractal pattern called the Sierpinski triangle [18, 24]. We present our macrotile and "tile gadget" design that allows for the type of inter-macrotile cooperativity required for algorithmic growth, then show the results of kSAM simulations of that system implemented with varying levels of cooperativity and intra-macrotile slat counts. The results are very promising and show that growth can be sped up greatly by decreasing slat type counts while also remaining essentially error-free within relatively broad ranges of parameters in comparison with previously designed algorithmic systems.

The organization of this paper is as follows. In Section 2 we define the aSAM and in Section 3 we define the kSAM and discuss various properties of it and how we tuned parameters for our simulations. In Section 4 we present our first method for reducing slat type

counts and kSAM simulation results for systems designed using that method. In Section 5 we present our general design for algorithmic systems and a specific design of a system to generate the Sierpinski triangle pattern, then kSAM simulation results of it. Finally, in Section 6 we summarize our results and discuss future directions.

2 The abstract Slat Assembly Model

In this section we briefly introduce the abstract Slat Assembly Model. The abstract Slat Assembly Model (aSAM) is a generalization of the aTAM [27] in which the fundamental components are *slats*, $n \times 1 \times 1$ polyominoes made of cubes in 3D space. Similar to tiles, slats can have *glues* (also referred to as *handles*) on each of their $4n + 2$ faces. Each glue is identified by a *label*, some string of characters (or sometimes a color), and a non-negative integer *strength*. Each glue has a complementary glue which shares its strength. In this paper we will often denote complementary glues using the same labels but with one appended by an asterisk (e.g. "label" and "label*"). Furthermore, we make a distinction between *slats* and *slat types*, the latter being just a description of the glues and length of a slat with no defined position or orientation. The position and orientation of slats is restricted to the 3D integer lattice and two slats which sit incident to one another are said to be *attached* or *bound* with strength s if they share complementary glues of strength s on their abutting faces. An *assembly* is simply a set of non-overlapping slats.

A *slat assembly system* (SAS) consists of a finite set of slat types, an assembly called the *seed assembly* which acts as the starting point for growth, and a positive integer called the *binding threshold*. The binding threshold describes the minimum cumulative glue strength needed for a slat to stably attach to a growing assembly. Growth in the aSAM is described by a sequence of slat attachments. Any slat which could sit on the perimeter of an assembly so that it would be attached to other slats with a cumulative strength meeting the binding threshold is a candidate for attachment, and attachments are assumed to happen non-deterministically. Any assembly which could result from a sequence of slat attachments beginning with the seed assembly of a SAS $\mathcal{S}$ and using only those slat types in the slat set of $\mathcal{S}$ is said to be *producible* in $\mathcal{S}$. Any assembly which permits no additional slat attachments is called *terminal*.

The aSAM is an idealized model intended to abstractly describe the growth of DNA-based slats under ideal conditions, though it makes no attempt to model realist growth dynamics. Rather, the aSAM is useful for designing and understanding complex slat systems on a logical level rather than a physical one. Considering slat systems in the framework of the aSAM allows us to investigate questions such as how many unique handles/glues are necessary to perform a desired task or how many handles could an erroneous slat bind with at any given time. Furthermore, the discrete nature of the aSAM is ideal for computer simulation.

We also consider a restricted version of the aSAM which we call the aSAM$^-$ which limits slat attachments in several ways. These restrictions include slats only being allowed to attach in the planes $z = 0$ and $z = 1$, and the requirement that any two slats sharing the same type will always attach in the same orientation. These restrictions exist to limit our designs to those which grow in a similar manner to those slat system in [28], but also allow for more efficient computer simulation.

2.1 SlatTAS: an aSAM$^-$ simulator

We have developed and freely released the source code for a Python-based graphical simulator for the aSAM$^-$ (and kSAM, see Section 3) called SlatTAS. It can be downloaded from `self-assembly.net` via a link on the page here [15].

2.2 Slat system design parameters

In this section, we provide an example of a SAS and some related terminology that will be used throughout the remaining sections.

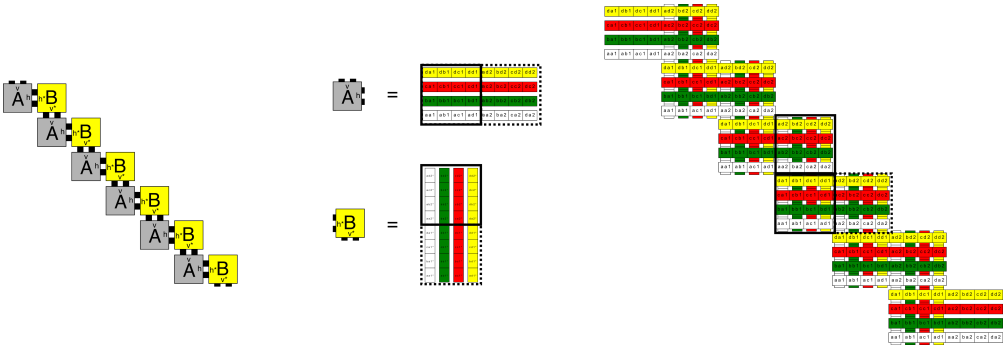


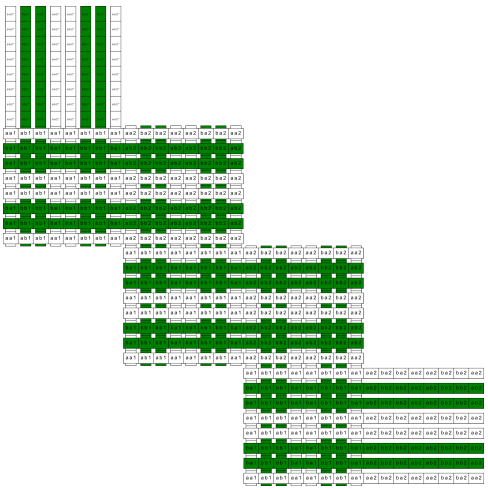
Figure 3 (left) A portion of an example "ribbon" made by a repeating pattern of 2 square tiles, "A" and "B", where one copy of A is used as the seed. (Note that the ribbon would extend infinitely in both directions.) (middle) An example representation of tiles "A" and "B" as slats using macrotiles of size 4. The outlines show the macrotile locations occupied by the slats. (right) A portion of an example system of slats simulating this tile system.

The number of slats to which an incoming slat must bind (since all glues are of strength 1) is equal to the parameter τ and is called the *cooperativity* of a system (a.k.a., *coordination number*, the term used in [21, 28]). Similar to the vast majority of the designs of [28], our systems will be designed so that slats assemble in arrangements which we refer to as forming *macrotiles*. Given a system whose cooperativity is c, a macrotile is simply a $c \times c$ square through which c horizontal slats, and c vertical slats, extend and bind to each other. Multiple macrotiles may be logically combined to form a *tile gadget* (or simply *gadget*) which can be interpreted as a group of slats that "work together" to perform a logical operation, which is often analogous to a tile in an aTAM system that we have converted into a functionally equivalent slat system. For example, the set of four slats in Figure 3 which are shown to "simulate" tile "A" can be thought of as a (relatively) simple gadget whose slats occupy two macrotiles. The macrotile in which the slats initially bind can be thought of as providing the input, and once each slat binds to the four slats in the input macrotile, it extends into the output macrotile. Once all four slats have bound to the input slats, they provide enough domains in each of four horizontal rows for the slats of the gadget simulating the "B" tile attach. This process can alternate between horizontal and vertical gadget attachment infinitely many times, in both the up-left and down-right directions.

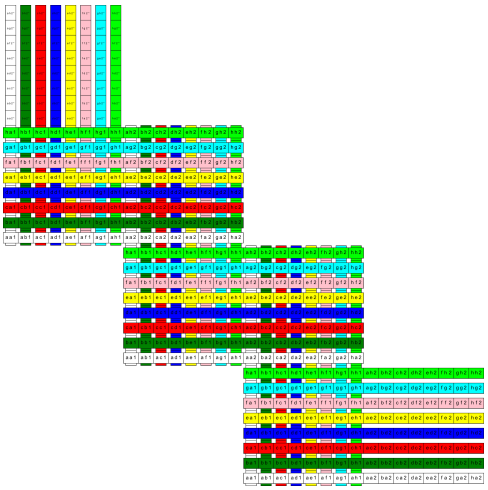
The macrotiles of a gadget can serve as *input*, *output*, and *connector* (to be seen later) macrotiles. We define the *slat count* as the count of unique slat types in each layer of a macrotile. Figures 4a and 4b show example ribbons with cooperativity 8 and slat counts 8 and 2, respectively. We use the term *motif* to refer to the pattern of slat types in a macrotile. For instance, in Figure 4a if the white vertical slats are of type w and the green are of type g, we could denote the motif as $wggwwggw$.

3 The kinetic Slat Assembly Model

To better understand how crisscross slat assemblies grow, we extend the framework of the kinetic Tile-Assembly Model (kTAM) to incorporate slats instead of square tiles. In the kTAM [27], tile attachments are modeled as reversible processes with forward and reverse



(a) Slat count 2 (i.e. each macrotile contains slats of only two different types in each direction).



(b) Slat count 8 (i.e. each slat of each direction in a macrotile is unique).

Figure 4 Example slat ribbons with macrotile size 8 (i.e. cooperativity 8) and varying slat count. Slats running in different directions are of different types, and for each direction each type has a unique color.

rates. Furthermore, the kTAM models *seeded* growth, beginning from a predefined *seed* assembly which is assumed to exist at lower concentrations than the individual tiles. In the kTAM, the forward rate of attachment for tile t has the form $r_f = k_f[t]$, where k_f is the reaction rate constant and $[t]$ is the concentration of t. The kTAM assumes that the rate of tile detachment depends primarily on the number of glue bonds formed by each tile. A tile with only 1 bound glue, for instance, should detach much more quickly than a tile with 2 or more bound glues. The reverse rate therefore depends on the free energy of a typical bond which is denoted ΔG°_{se} where se stands for *sticky end*. Consequently, the reverse rate describing the detachment of a tile with b glue bonds has the form

$$r_{r,b} = k_f u_0 \exp\left(b\frac{\Delta G^\circ_{se}}{RT} + \alpha\right)$$

where u_0 is a standard reference concentration, T is the temperature in Kelvin, R is the molar gas constant, and α is a unitless free energy parameter associated with other factors specific to a particular realization of the tiles. These rates are generally translated into a more symmetric form [10] by defining $\hat{k_f} \stackrel{\text{def}}{=} u_0 k_f \exp(\alpha)$, $G_{mc} \stackrel{\text{def}}{=} \alpha - \log(\frac{[t]}{u_0})$, and $G_{se} \stackrel{\text{def}}{=} -\frac{\Delta G^\circ_{se}}{RT}$. These can then be substituted into the original rate formula to get the following.

$$r_f = \hat{k_f}\exp(-G_{mc}) \qquad\qquad r_{r,b} = \hat{k_f}\exp(-bG_{se})$$

In this form, the parameter G_{mc} describes tile concentrations logarithmically with larger values corresponding to smaller concentrations, and G_{se} describes the free energy of a strength 1 glue with larger values corresponding to stronger bonds. In the kTAM, the ratio G_{mc}/G_{se} plays an important role, analogous to the binding threshold τ in the aTAM. When this ratio is slightly less than 2 for instance, growth in the kTAM proceeds much like it would in a binding threshold 2 aTAM system with the same tiles. This is because at this ratio, tiles bound with strength 2 dissociate at a rate just barely less than tiles attach to the assembly while tiles bound with strength 1 or 0 dissociate with a significantly higher rate.

Consequently, a correctly bound tile (matching 2 glues) is likely to remain attached to the assembly for long enough for additional tile attachments to occur, matching glues with the correct tile and subsequently increasing its number of bonds so that it is held stably to the attachment. Incorrect tiles (matching 1 glue or fewer) on the other hand detach much more rapidly and very likely before any additional tile attachments occur. When this ratio is exactly equal to 2, or more generally the cooperativity of the tile system, then the kTAM describes how the system behaves at the melting temperature, i.e. even the strongest bound tiles detach just as often as tiles attach to the assembly. In general, the ratio of G_{mc} to G_{se} is often more pertinent when describing the expected dynamics of a system than the values of the individual parameters, though the parameters themselves are necessary for computer simulation of the kTAM. For a more detailed description of the kTAM and subsequent analyses for square tiles, see [10, 27].

Despite making several simplifying assumptions, the kTAM has been broadly successful in realistically describing growth and error phenomena in a variety of tile-based schemes of DNA self-asssembly [6, 9]. The model captures many critical features of physical tiles while remaining simple enough for meaningful mathematical manipulation and efficient computer simulation using the Gillespie algorithm [12]. Generalizing the kTAM to incorporate slats rather than square tiles is, in principle, as straightforward as it sounds. Instead of square tiles which occupy only a single grid location and contain at most 4 glues, we consider slats which can occupy several adjacent grid locations and have a number of glues proportional to their length. For purposes of brevity and clarity, we will refer to this generalization as the *kinetic Slat Assembly Model* or kSAM to distinguish it from the kTAM for square tiles.

3.1 Finding appropriate parameter values for physical slats

While many important properties of a kSAM system can be inferred from the ratio G_{mc}/G_{se}, it's useful to find specific values for these individual parameters which yield simulations with accurate growth rates. Other than α which describes an entropic cost associated with a specific implementation of slats, all of the factors defining the parameters G_{mc}, G_{se}, and $\hat{k_f}$ such as temperature and slat concentrations are generally known. We can roughly estimate the value of α by noting that, for slat systems with cooperativity c, the melting temperature should correspond to values for G_{mc} and G_{se} such that $G_{mc} = cG_{se}$. Consequently, by substituting G_{mc} and G_{se} with their definitions we find that

$$\alpha - \log\left(\frac{[s]}{u_0}\right) = -c\frac{\Delta G^\circ_{se}}{RT}.$$

Furthermore, $\Delta G^\circ_{se} = \Delta H^\circ_{se} - T\Delta S^\circ_{se}$ where ΔH°_{se} and ΔS°_{se} are the enthalpy and entropy associated with a single sticky end bond whose values can be approximated using the nearest neighbor model. Rearranging the above equation, thus yields the following expression for α which assumes that T is the melting temperature of the system.

$$\alpha = \frac{c}{R}\left(\Delta S^\circ_{se} - \frac{\Delta H^\circ_{se}}{T}\right) + \log\left(\frac{[s]}{u_0}\right)$$

In [28], the authors determined the melting temperature for origami slat ribbons using cooperativity values of 8 and 16 and using handle lengths of 6, 7, and 8 nucleotides. Using their values for melting temperature and the corresponding slat concentrations, and applying the nearest neighbor model to their handle sequences therefore allows us to find a value for α. Using the handle sequences from [28] we calculated typical values for 7nt handles to be about $\Delta H^\circ_{se} \approx -47\text{kcal}\,\text{mol}^{-1}$ and $\Delta S^\circ_{se} \approx -142\text{cal}\,\text{mol}^{-1}\,\text{K}^{-1}$. For their slat concentration

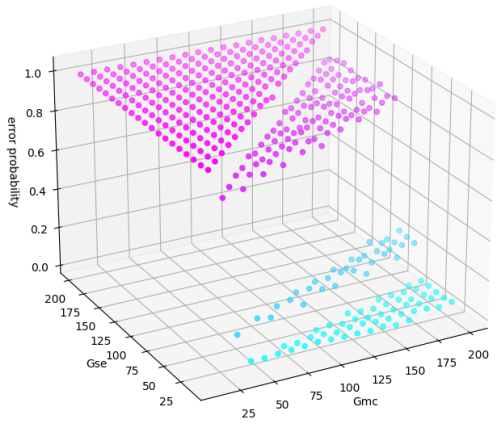


Figure 5 Error rates for kinetic simulations of cooperativity 16 ribbons at G_{mc} and G_{se} values ranging from 10 to 200. The general shape of the error curve matches the trends predicted by the kinetic trapping model with sharp jumps as the ratio G_{mc}/G_{se} crosses an integer value. Furthermore, when $G_{mc} < G_{se}$ the error probability is 1.

of 20nM and melting temperature of 42.1°C = 315.25K, these values yield a value for α of around 39. Admittedly, this approximation isn't perfect; for one, the formula for α is quite sensitive to the values of ΔH°_{se} and ΔS°_{se} which can vary a bit for the same glue sequence depending on the specific duplex table used in the nearest neighbor model calculations. For our simulations we chose to use $\alpha = 40$, though we note that for our purposes the specific value for α makes little difference in the error dynamics of our simulated slat systems.

3.2 A kSAM simulator

We have developed and freely released the source code for a C++ based stochastic simulator for both the aSAM and kSAM, which has been highly optimized for faster simulation than the existing SlatTAS [15]. It can be downloaded from `self-assembly.net` via a link on the page here [13]. More generally, our simulator is capable of handling arbitrary 3D polyomino-shaped tiles (including square/cube tiles) and can be configured to simulate 2D and almost-3D (with only 2 layers in the z-axis) systems as well by use of a dimension restriction which allows for specifying minimum and maximum allowed coordinates in all 3 dimensions if desired. The simulator can also use multiple threads to simulate a system ensemble. All kinetic simulations in this paper were performed using our simulator in kinetic mode and our error metrics were calculated using reference assemblies generated by our simulator in abstract mode.

4 Optimized implementation of crisscross ribbons via slat type reuse

In this section, we describe our first technique for reducing slat type counts, which involves slat type reuse within macrotiles. We then describe the range of systems with varying amounts of such intra-macrotile slat reuse that we designed and then tested via kSAM simulations, and discuss the results of those simulations.

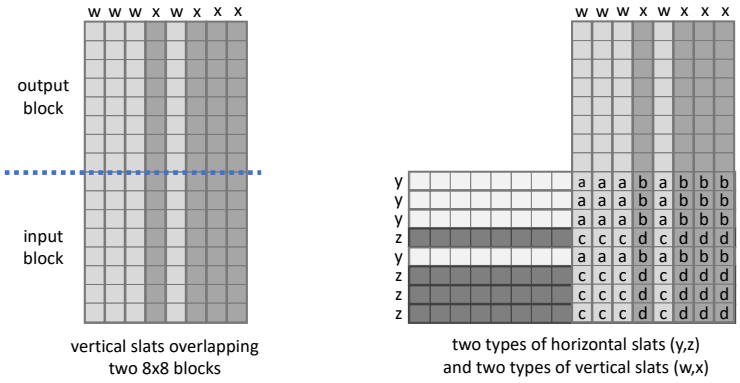


Figure 6 Example using two types of slats in each direction for 8×8 macrotiles. (*left*): Two types of vertical slats, w and x, with four copies of each used for the eight total vertical slats used in the input and output macrotiles. (*right*): Two types of horizontal slats, y and z, where input glues of w and x bind to output glues of y and z. This illustrates how slat re-use within a macrotile forces re-use of glues. Where slats w and y overlap, they must have a complementary glue (a on w and a^* on y, latter not depicted since the output glues of the y slat are underneath the w and x slats). By symmetry, that same glue a is forced to be used wherever slats w and y intersect. In this situation, the maximum number of glues that can possibly be used is only four: a, b, c, d, increasing the potential for misaligned slats to have complementary glues in adjacent locations over systems which use larger numbers of glues.

4.1 Slat layout in macrotile designs

Most assembly schemes in [28] consider slat assembly in a modular way, by partitioning the integer lattice $\mathbb{Z}^2$ into $n \times n$ macrotiles. The n slats that assemble to fill in an $n \times n$ macrotile can be considered as "simulating" a square tile in the abstract and kinetic Tile Assembly Models [27]. In [28], a unique slat type was used in every one of the n relative positions within a macrotile (even for periodic structures such as infinite 1D ribbons and 2D sheets that used the rectangular macrotile growth pattern and repeated the same set of slats in different macrotiles). However, we have analyzed patterns of possible slat reuse within macrotiles and developed a technique for greatly reducing how many are required in each while seeking to maintain (mostly) error-free growth, which we now describe.

Figure 6 shows an example of using only two types of slats in each direction when we use cooperativity 8 to have 8×8 macrotiles. Its caption explains how this slat reuse forces the reuse of glues. This reuse of glues in turn means we must confront the possibility that slats can bind "strongly" somewhere that they should not (perhaps flipped and/or translated such that they are not in alignment with macrotile boundaries) due to complementarity of a *partial* subset of glues. Figure 7 shows some potential scenarios with erroneous binding. Although we declare in the abstract aSAM model that no binding occurs if the number of matching glues is less than the cooperativity value n, in the kSAM and in actual experiments it is possible for such situations to allow for slat bindings with "close" to n bonds (although with expected duration of attachment to diminish as the distance from n increases). Therefore, we strive to ensure that slat and glue patterns are designed so that the maximum strength of any such incorrect slat binding is minimized. We analyze motif patterns to detect and minimize a property we call *auto-correlation* in order to minimize the errors that occur during self-assembly.

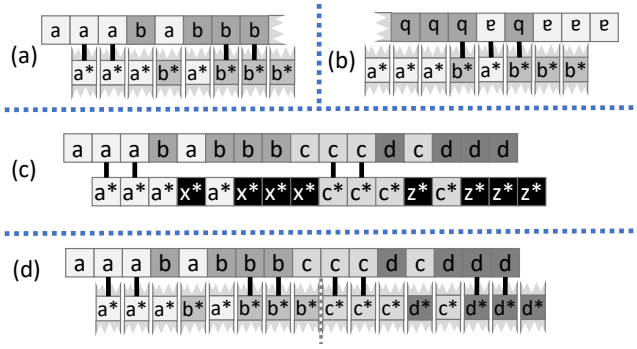


Figure 7 Design of slat arrangement to minimize overlap of complementary glues in unintended slat bindings. (a): The input glues of a horizontal slat (*aaababbb*; output glues on right not depicted) binding with incorrect translation to the output glues of eight vertical slats presenting glues $a^*a^*a^*b^*a^*b^*b^*b^*$. This arrangement of input glues has *auto-correlation* 4: this translation makes 4 of the glues bind, and every other nonzero translation also has at most 4 matching glues. This means the slat could erroneously bind with strength 4 in the incorrect x position. (b): A string of input glues for a slat with reversed auto-correlation 3. Such a slat could bind with incorrect 180 degree rotation with strength 3. (c): Two full slats binding erroneously to each other. A horizontal slat (on top; input glues a, b on left, output glues c, d on right) and a vertical slat (on bottom; output glues a^*, x^* on left, input glues c^*, z^* on right) bind. The potential for such binding occurs in situations such as the repeating ribbon of Figure 4a, where (in the normal expected binding) a horizontal slat type binds to a vertical slat type, then the same vertical slat type binds to the horizontal slat. By our glue design; in this scenario at most one glue can be shared on each side, e.g., since they share a on the left, they have different glues (b vs. x) for the other type on the left. This check was done in [28]. (d): Similar to part (a), but considers output glues of a slat also. In this example, the left-side block are input glues for the horizontal slat, and the right-side block are the outputs. This describes the scenario where, after vertical slats begin binding on the right (with less than total cooperativity) to horizontal slats (not shown), before the block is complete with horizontal slats, a correct horizontal slat with incorrect x-translation "backfills" in the block.

4.2 Testing slat reuse via kinetic simulations

In [28], the authors, among several other experiments, grew a variety of ribbons using DNA-origami crisscross slats and investigated how their growth was affected by the number of unique slat types used. Specifically, these ribbons, all of which used a cooperativity value of 16, were designed to use either 8, 16, 32, or 64 unique slat types in each of the two layers of growth, attaching periodically. They evaluated the growth of these ribbons both in the situation where the concentration of individual slats was kept constant, and where the total concentration of all slats was held constant (i.e. smaller slat counts having higher per-slat concentrations). While the growth of these ribbons was generally similar when individual slat concentrations were maintained, they did observe a significant decrease in growth as slat count increased when total slat concentrations were fixed.

Here we explore the effects of slat count on ribbon growth conversely, to determine the extent to which we can expect to decrease slat counts while preserving rapid correct growth of the ribbons. To do this, we designed several cooperativity 16 ribbon systems with decreasing slat count and evaluated their growth dynamics within the framework of the kSAM. Our systems included ribbons with 2, 4, 8, and 16 unique slat types for each layer and we performed kSAM simulations of these systems both with individual slat concentrations held constant and with total slat concentrations held constant. The ribbon systems we used are illustrated in Figure 8 and additional info can be seen in Table 1. We chose to implement zig-zag ribbons which more closely match our motif for algorithmic growth than the "staggered" ribbons of some of the designs in [28].

We simulated each of these systems at a fixed value of G_{mc} and varied G_{se}. In principle, this corresponds to growing the slat systems with fixed slat concentrations at a variety of temperatures. The specific value of G_{mc} used was 58 which corresponds to a slat concentration

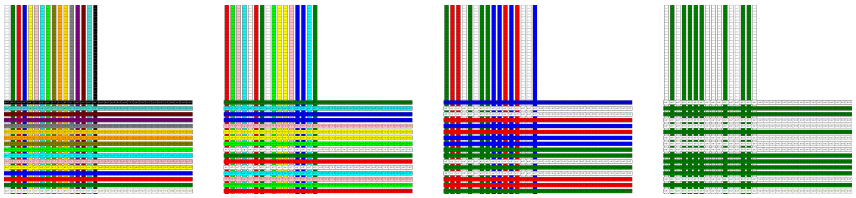


Figure 8 The cooperativity 16 ribbons simulated in the kSAM with various slat counts. Slat counts from left to right: 16, 8, 4, 2.

Table 1 For the cooperativity 16 ribbon systems tested via kSAM simulations with given slat counts, the motifs, auto-correlations and cross-correlations.

Slat count	Motif	Auto-correlation	Cross-correlation
16	`abcdefghijklmnop`	1	2
8	`chfgacbaheefddgb`	2	4
4	`bccababbddcdcaad`	4	4
2	`ababbbbaaabaabba`	8	8

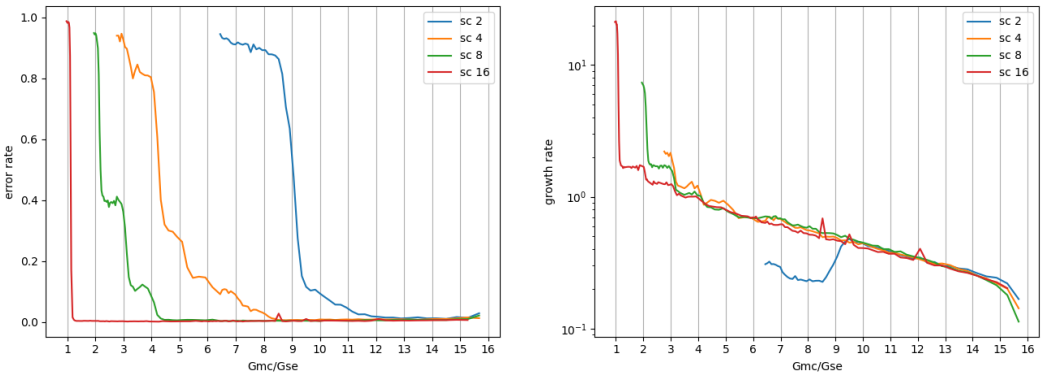
of 15nM using the value for α derived in Section 3.1. Values of G_{se} were chosen so that G_{mc}/G_{se} values spanned the range from 1 to 16. For each ribbon, we wanted to simulate 2 situations: (1) with individual slat counts kept constant at 15nM for all ribbons, and (2) with total slat concentrations held constant at 480nM across all ribbons. To keep total slat concentrations fixed, systems with smaller slat counts had their slat concentrations adjusted[1]. Note that for both the fixed individual slat concentrations and fixed total concentrations, concentrations are identical for the slat count 16 ribbons, so those were only simulated once.

To estimate the rate of erroneous slat attachment, we additionally simulated each system in the error-free aSAM with a binding threshold of 16. This resulted in a reference assembly containing only correctly attached slats to which our kinetic simulations could be compared. We then consider a slat to be correct in the kinetic simulation when a corresponding slat, of the same type in the same translation, exists in the reference assembly. To estimate growth rates in our simulated systems, we note that the time between events during a kinetic simulation can be determined by sampling a value of Δt from an exponential distribution whose rate parameter is the net rate of any event (attachment or detachment) occurring [27]. The growth time of a simulation can thus be calculated by summing the sampled Δt values for each event which occurs in the simulation. Growth rate is then simply estimated as the number of slats which attached by the total growth time.

4.3 Results for ribbons simulated with fixed individual slat concentrations

Figures 9a and 9b describe the estimated error and growth rates of the various ribbons with a fixed individual slat concentration. Note that $G_{mc}/G_{se} = 16$ corresponds to the melting temperature since with those parameters, the rate of slat attachment is equal to the rate of detachment of slats with 16 bound glues. Consequently, the growth rate beyond that point will be 0. Near the melting temperature, all ribbons exhibit very few erroneous

[1] In our simulation code, this is implemented as a multiplier to the forward rate, rather than adjusting G_{mc}, though the effect is the same.



(a) Simulated error rates of our cooperativity 16 ribbons when individual slat concentrations are held constant.

(b) Simulated growth rates of our cooperativity 16 ribbons on a log scale when individual slat concentrations are held constant.

Figure 9 Results for kinetic simulations of cooperativity 16 ribbons with fixed individual slat concentrations. Results represent averages over 100 simulations.

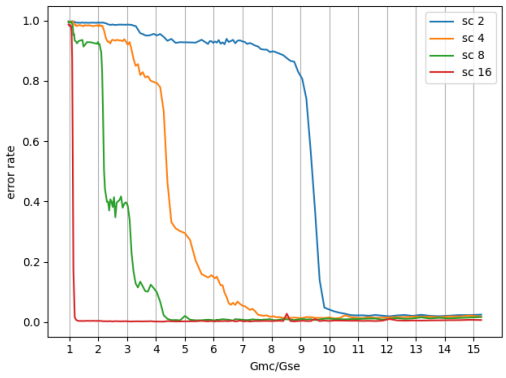
slat attachments. Interestingly, the error rate for all ribbons remain essentially at 0 for $G_{mc}/G_{se} > 12$. Additionally, growth rates for all ribbons in this range are essentially equivalent between the differening slat counts. Together these results suggest that there exists a range of temperatures in which the slat count 2 ribbons grow just as well as the slat count 4 and 8 ribbons. As discussed further in Section 7.1 of the appendix, the growth rate of the slat count 2 ribbons interestingly decreases when G_{mc}/G_{se} drows below around 9. This seems to be due to a high frequency of erroneous slat attachments which quickly occupy most available stable attachment sites preventing further growth.

In general, the error rates for the systems drastically increase as the G_{mc}/G_{se} ratio drops to the value corresponding to the auto-correlation values of the systems. This is because for each system there exist one or more slats that can bind while in an incorrect/misaligned location with a number of glues equal to that system's auto-correlation value, and when that is near or equal G_{mc}/G_{se}, those attachments are relatively stable. Thus, the prevalence of errors rapidly increases near those values. Due to space constraints, more details of the types of errors observed can be found in Section 7.1 of the appendix.

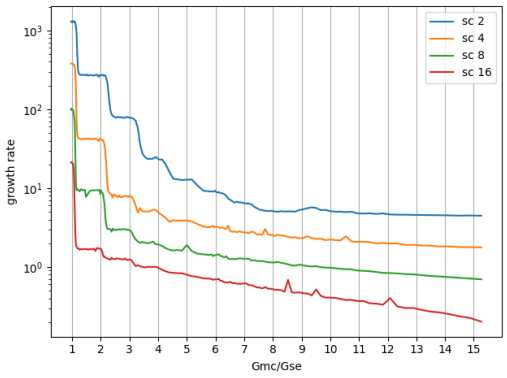
The results of these kSAM simulations imply that there are ranges of G_{mc} and G_{se} where ribbon growth can remain almost entirely error-free even when the slat counts are reduced to the nearly optimal value of 2. (Note that any system using cooperativity n with a slat count of 1 would necessarily have auto-correlation value of $n-1$, and thus a very narrow possible range for G_{mc}/G_{se} with low errors.) Additionally, even with individual slat concentrations held constant across systems, the growth rates were roughly equivalent across wide ranges of G_{mc}/G_{se}, meaning that the lower absolute concentrations of slats (and thus lower arrival rates of slats at frontier locations) was balanced by the higher likelihood of a slat type being the correct type for a location in which it randomly arrives.

4.4 Results for ribbons simulated with fixed total slat concentrations

In addition to simulating the ribbons with fixed individual slat concentrations, we also simulated the ribbons with total concentrations fixed. That is, systems using fewer unique slats had higher concentrations for each slat. For these experiments, we used the same slat designs as in the previous section (with details shown in Table 1), changing only slat



(a) Error rates for cooperativity 16 ribbons using slat counts of 2, 4, 8, and 16 using a fixed total concentration of all slats in each system. Notice that despite having different concentrations, the results are near identical to those in Figure 9a.



(b) Growth rates for cooperativity 16 ribbons using slat counts of 2, 4, 8, and 16 with fixed total concentration of all tiles in each system. Results are plotted on a log scale. Growth rates are larger for smaller slat counts suggesting that decreasing slat counts allows for faster growth of ribbons.

Figure 10 Results for kinetic simulations of cooperativity 16 ribbons with fixed total slat concentrations. Results are averaged over 100 simulations.

concentrations. This is in line with the ribbon experiments performed in [28]. Error rates and growth rates are calculated using the same method for the analogous results with fixed individual slat counts. Our results are summarized in Figure 10a and Figure 10b. While decreasing slat counts caused ribbons to have narrower regions of correct growth, our results show that under the right conditions, all ribbons were able to grow essentially error free. Moreover, in our simulations, halving slat counts lead to growth rates increasing by an average factor of 2.35 at $G_{mc}/G_{se} = 11$. In fact over the range of G_{mc}/G_{se} values where all ribbons grew with little error, (i.e. above 11), the ratio of growth rates between ribbons whose slat counts differed by a factor of 2 was consistently greater than 2 as illustrated in Figure 19. In other words, halving slat counts more than doubled growth rates in our simulations. Whether these results would translate into the lab is unclear, but regardless our results suggest that decreasing slat counts can be a powerful tool for improving growth rates without sacrificing much error. Importantly, this technique for reducing slat type counts is applicable in general, to all designs using macrotiles. (For more detailed examination of some of the errors observed, please see Section 7.2.)

5 Algorithmic self-assembly using crisscross slats

An $n \times n$ square composed of square tiles in the aTAM can self-assemble in a system which uses a unique tile type at each location (a so-called *fully-addressable* or *hard-coded* system), requiring n^2 unique tile types. However, an algorithmic system can form a structure of the exact same shape using an optimal (via an information theoretic argument) $\frac{\log(n)}{\log(\log(n))}$ tile types [1, 25], meaning that the hard-coded system uses exponentially more tile types than the algorithmic system. As previously mentioned, reducing the number of unique component types has the benefits of making a physical implementation via DNA (1) cheaper, (2) faster, and (3) require fewer unique domains thus making their individual binding characteristics more uniform. However, in order to leverage the power of algorithmic self-assembly, a slightly different notion of cooperativity than previously discussed is necessary. While the previously

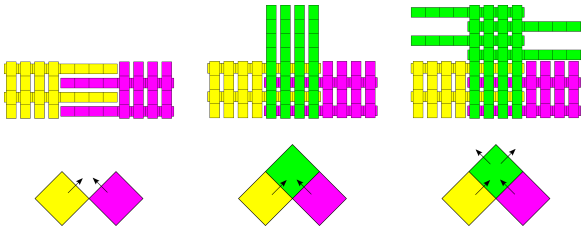


Figure 11 An example of "algorithmic" cooperation, i.e. the inputs of one gadget's slats are the outputs of slats from two different gadgets. In this case, the yellow slats belong to one gadget and the pink to another. They overlap in the location of the central macrotile to provide input to the green slats. This is often referred to as *across-the-gap* cooperation. This process is analogous to the algorithmic cooperation realized by square yellow and pink tiles cooperatively binding to a square green tile.

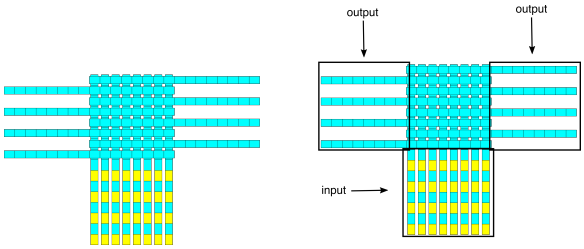


Figure 12 An example gadget that receives its input in a macrotile at the bottom, then has slats which propagate output to macrotiles to the top left and right.

discussed notion of cooperativity concerned the binding of a single slat to multiple other (orthogonally oriented) slats in order to attach to an assembly, in the examples shown (e.g. Figure 3) all slats bound to as input for one gadget were acting as output for a single other gadget. This effectively provides only a single logical input to direct the growth of each gadget. Algorithmic growth, on the other hand, requires that some gadgets receive input from at least two distinct gadgets, allowing the combined information from both input gadgets to direct the growth of the new gadget. (This was shown to be necessary by the requirement of a minimum temperature value of 2 in the aTAM [19, 20].) An example of such algorithmic cooperativity implemented via slats in 4×4 macrotiles can be seen in Figure 11.

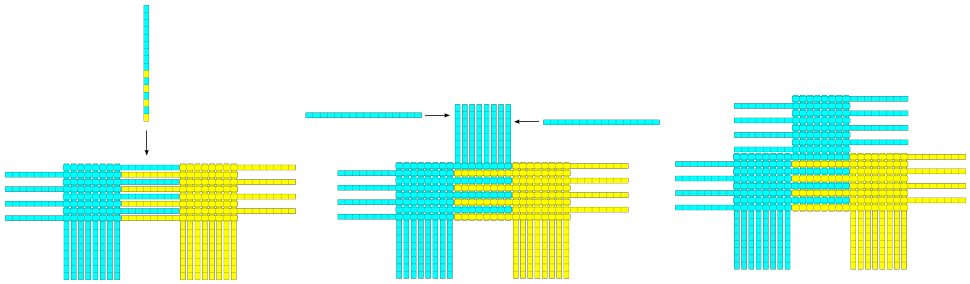


Figure 13 Order of growth of a gadget using algorithmic cooperation.

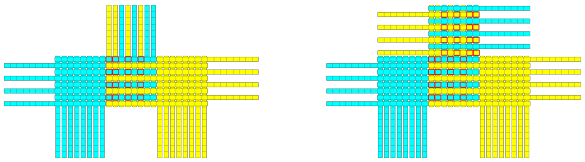
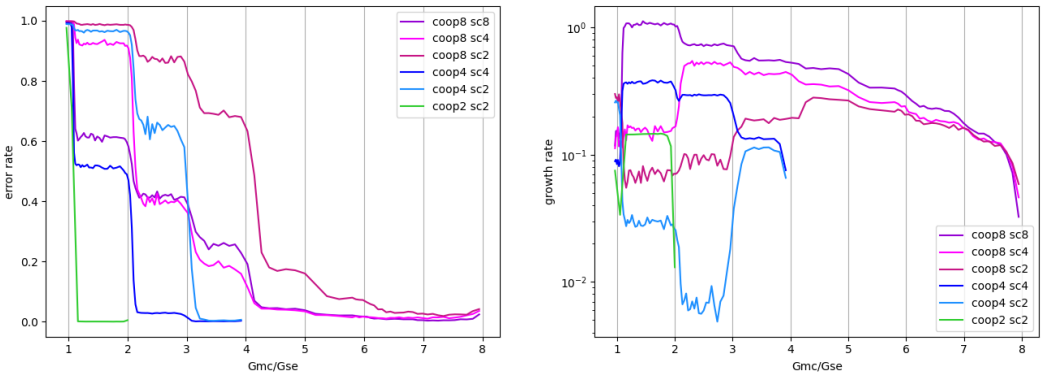


Figure 14 Example of algorithmic error propagation through a gadget.

5.1 Kinetic simulations and error analyses of algorithmic systems

We hypothesized that our design for algorithmic systems using slats arranged in gadgets composed of macrotiles should exhibit strong proofreading characteristics. The reasoning for this is depicted in Figures 12–14 where (1) the input and output regions of a gadget are depicted (Figure 12), (2) the ordering of correct growth is shown (Figure 13), and (3) the number of concurrent errors that would need to occur and persist in order for slats to bind with even half of the designed binding strength in order to propagate algorithmic errors is exemplified Figure 14. In general, for a macrotile scheme at cooperativity k (i.e. using slats of length $2k$), at least k erroneous half-strength slat attachments are required for at least half of a macrotile's outputs to encode the wrong value. While, this is the same number of erroneous attachments necessary to invalidate a square tile system using $k \times k$ block-replacement, our macrotiles consist of only k slats rather than k^2 square tiles. In other words, a square tile block-replacement scheme requires a quadratic increase in tile complexity, compared to a linear increase for our slat based macrotiles, to achieve the same amount of proofreading.

The discrete self-similar fractal pattern known as the Sierpinski triangle is a well-studied pattern in aTAM and kTAM self-assembly, both theoretically [18] and experimentally [24]. This is due to its relatively simple algorithmic logic (each location represents a 0 or 1 bit value that is the `xor` of the bits of its two input neighbors) that results in an infinite, aperiodic pattern known as a discrete self-similar fractal. (See Section 7.3 for depictions of the Sierpinski triangle.) We designed several macrotile-based algorithmic slat systems which grow to form a Sierpinski triangle pattern using cooperativity values of 2, 4, and 8. In total we simulated 6 designs, permuting cooperativity and slat counts. For cooperativity 8 triangles we tested slat counts of 2, 4, and 8; for cooperativity 4 triangles we tested slat counts of 2 and 4, and our cooperativity 2 triangles used a slat count of 2. Concentrations of individual slats was kept constant at 15nM per slat. The results of the simulations are summerized in Figure 15a. Our results show that while decreasing slat counts increased error rates consistently, there were still ranges of conditions under which all triangles grew with little error. Interestingly, for the cooperativity 4 triangles, using a slat count of 2, growth rates slowed significantly below a G_{mc}/G_{se} of 3. Much like the slat count 2 ribbons in Section 4, we suspect that this is due to erroneous slats quickly filling up most of the stable attachment sites preventing further growth. This is supported by the fact that growth rates increase below a G_{mc}/G_{se} of 2 where stable attachment sites require fewer correct glues. Regardless, our triangle systems exhibit a remarkable tolerance to infrequent errors as can be seen in Figure 23 and Figure 24. These results support the idea that even for algorithmic systems, under ideal conditions, reducing slat counts is possible without sacrificing much in terms of error.



(a) Rate of erroneous slat attachment in kinetic simulations of our algorithmic slat systems as a function of G_{mc}/G_{se}. All systems exibited little error in conditions near their respective melting points, where G_{mc}/G_{se} equals the cooperativity value of the system.

(b) Log growth rate during kinetic simulations of our algorithmic slat systems as a function of G_{mc}/G_{se}. Here growth rate is calculated as the number of slat additions per unit of simulated time.

Figure 15 Results for kinetic simulations of Sierpinski triangle systems. Each data point represents the average of 100 simulations.

6 Conclusions and Future Work

In this paper, we have introduced an abstract mathematical model for self-assembling systems composed of slats that can combine in layers using high levels of cooperativity, and shown how to design algorithmic self-assembling systems of slats within it. We have also introduced a more physically realistic "kinetic" model capable of capturing many types of errors that occur in laboratory implementations of self-assembling systems, and showed that we were able to tune parameters of simulations within that model to match results of systems from [28]. We then presented a technique for reducing the number of unique slat types required to build ribbon structures (that generalizes to all macrotile-based designs) and showed via simulations that growth rates can be greatly accelerated while low error rates are maintained. Finally, we presented a technique for designing algorithmic self-assembling systems of slats and demonstrated it by designing a system that generates the Sierpinski triangle pattern, which we also simulated to show that extremely low error rates can be maintained. Due to the fact that algorithmic systems are capable of self-assembling structures while utilizing only a logarithmic number of unique components relative to hard-coded structures, and combined with the first technique for reducing slat types, our designs result in slat systems with dramatically fewer slat types than previous designs. Simulations show that these systems will therefore self-assemble much faster, and they will also be much easier and cheaper to implement.

We will be implementing these and similar systems using DNA origami slats and comparing the laboratory results to the results of our simulations, then updating the model (and our designs) as necessary to refine the designs. Future work could include new designs of macrotiles so that the inter-macrotile cooperativity needed for algorithmic self-assembly is achieved through different slat patterns and/or orientations, and then simulations and laboratory experiments to see which are best. Also, further examination of the types of errors that occur during simulations and laboratory experiments may result in additional checks and optimizations to be made for designs. It is our hope that crisscross slat based self-assembling

systems will achieve high-levels of algorithmic sophistication while maintaining low enough levels of algorithmic errors to realize much more of the theoretical potential that has been pursued for so long.

References

1 Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 740–748, Hersonissos, Greece, 2001. doi:10.1145/380752.380881.

2 Nathaniel Bryans, Ehsan Chiniforooshan, David Doty, Lila Kari, and Shinnosuke Seki. The power of nondeterminism in self-assembly. *Theory of Computing*, 9:1–29, 2013. doi:10.4086/toc.2013.v009a001.

3 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors): Self-assembly in the 2HAM vs. aTAM. In Natacha Portier and Thomas Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 172–184, 2013.

4 Cameron T. Chalk, Eric Martinez, Robert T. Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, 80(4):1383–1409, 2018. doi:10.1007/s00453-017-0318-0.

5 Ho-Lin Chen, David Doty, and Shinnosuke Seki. Program size and temperature in self-assembly. In *ISAAC 2011: Proceedings of the 22nd International Symposium on Algorithms and Computation*, volume 7074 of *Lecture Notes in Computer Science*, pages 445–453. Springer-Verlag, 2011.

6 Ho-Lin Chen, Rebecca Schulman, Ashish Goel, and Erik Winfree. Reducing facet nucleation during algorithmic self-assembly. *Nano Letters*, 7(9):2913–2919, 2007.

7 E. D. Demaine, M. L. Demaine, S. P. Fekete, M. J. Patitz, R. T. Schweller, A. Winslow, and D. Woods. One tile to rule them all: Simulating any tile assembly system with a single universal tile. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014),* IT University of Copenhagen, Denmark, July 8-11, 2014, volume 8572 of *LNCS*, pages 368–379, 2014.

8 David Doty, Jack H. Lutz, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, FOCS 2012, pages 302–310, 2012.

9 Constantine G Evans, Rizal F Hariadi, and Erik Winfree. Direct atomic force microscopy observation of dna tile crystal growth at the single-molecule level. *Journal of the American Chemical Society*, 134(25):10485–10492, 2012.

10 Constantine G Evans and Erik Winfree. Physical principles for DNA tile self-assembly. *Chemical Society Reviews*, 46(12):3808–3829, 2017.

11 David Furcy, Scott M Summers, and Christian Wendlandt. Self-assembly of and optimal encoding within thin rectangles at temperature-1 in 3d. *Theoretical Computer Science*, 872:55–78, 2021.

12 Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

13 Daniel Hader. RodSim: An Optimized Simulator for the kSAM. http://self-assembly.net/wiki/index.php?title=RodSim, 2023. [Online; accessed 28-April-2023].

14 Daniel Hader, Aaron Koch, Matthew J. Patitz, and Michael Sharp. The impacts of dimensionality, diffusion, and directedness on intrinsic universality in the abstract tile assembly model. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2607–2624. SIAM, 2020.

15 Daniel Hader and Matthew J. Patitz. SlatTAS: A Graphical Simulator for the aSAM$^-$. `http://self-assembly.net/wiki/index.php?title=SlatTAS`, 2023. [Online; accessed 28-April-2023].

16 Daniel Hader, Matthew J Patitz, and Scott M Summers. Fractal dimension of assemblies in the abstract tile assembly model. In *International Conference on Unconventional Computation and Natural Computation*, pages 116–130. Springer, 2021.

17 James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers. Computability and complexity in self-assembly. *Theory Comput. Syst.*, 48(3):617–647, 2011. `doi:10.1007/s00224-010-9252-0`.

18 James I. Lathrop, Jack H. Lutz, and Scott M. Summers. Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science*, 410:384–405, 2009.

19 Pierre-Étienne Meunier, Damien Regnault, and Damien Woods. The program-size complexity of self-assembled paths. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 727–737, 2020. `doi:10.1145/3357713.3384263`.

20 Pierre-Étienne Meunier and Damien Woods. The non-cooperative tile assembly model is not intrinsically universal or capable of bounded Turing machine simulation. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 328–341, 2017. `doi:10.1145/3055399.3055446`.

21 Dionis Minev, Christopher M Wintersinger, Anastasia Ershova, and William M Shih. Robust nucleation control via crisscross polymerization of highly coordinated DNA slats. *Nature Communications*, 12(1):1–9, 2021.

22 Matthew J. Patitz and Scott M. Summers. Self-assembly of decidable sets. *Natural Computing*, 10(2):853–877, 2011. `doi:10.1007/s11047-010-9218-9`.

23 Paul W. K. Rothemund. *Theory and Experiments in Algorithmic Self-Assembly*. PhD thesis, University of Southern California, December 2001.

24 Paul W. K Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biol*, 2(12):e424, December 2004.

25 Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstract). In *STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing*, pages 459–468, Portland, Oregon, United States, 2000. ACM.

26 David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. `doi:10.1137/S0097539704446712`.

27 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

28 Christopher M Wintersinger, Dionis Minev, Anastasia Ershova, Hiroshi M Sasaki, Gokul Gowri, Jonathan F Berengut, F Eduardo Corea-Dilbert, Peng Yin, and William M Shih. Multi-micron crisscross structures grown from dna-origami slats. *Nature Nanotechnology*, pages 1–9, 2022.

7 Appendix

This section contains technical details of results omitted from the main body of the paper due to space constraints.

7.1 Details regarding ribbon systems with fixed individual slat concentrations

Here we provide more in-depth analysis of the types and causes of errors that we observed in the simulations of ribbons with varying amount of slat reuse in Section 4.3. We attempt to

give our intuitions for why we think error and growth rates trend as they do. For example, the auto-correlation value for the slat count 2 system is 8, and for that system error rates begin to increase slowly as G_{mc}/G_{se} drops below about 12 until around 9.5 where it suddenly begins to spike. However, and importantly for continued growth of the ribbons, while the error rate in this region is distinctly larger than 0, we note that the errors present are generally isolated and do little to affect the overall growth of the ribbon. Figure 16 depicts a section taken from a typical slat count 2 ribbon at $G_{mc}/G_{se} = 10$. Erroneous slats can be seen clearly in the assembly as they are generally not aligned with the others, but because only a small number are present in each block, the further attachment of correct slats is not impeded significantly.

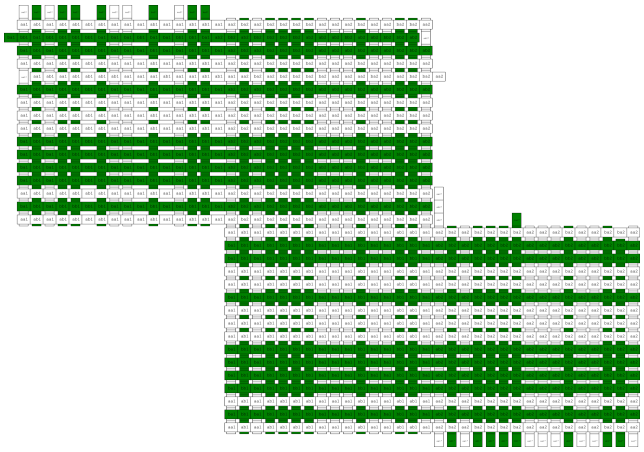
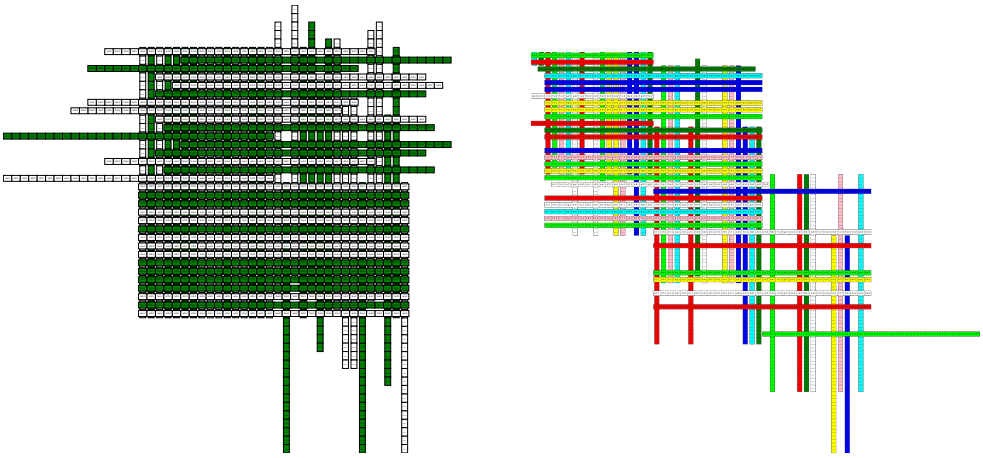


Figure 16 A typical section from ribbons with slat count 2 at $G_{mc}/G_{se} = 10$.

For both the slat count 4 and 8 ribbons, overall growth rates decrease monotonically with G_{mc}/G_{se} which is to be expected as higher values of G_{mc}/G_{se} correspond to higher temperatures. Surprisingly however, the slat count 2 ribbons seem to exhibit significantly reduced growth rates as G_{mc}/G_{se} decreases below about 9.5. Given that this region corresponds to growth with a significant amount of errors, this suggests, that the erroneous slat attachments eventually accumulate to the point where no further attachments are possible. Indeed, when we look at the assemblies resulting from these simulations, we find that this is the case as depicted in Figure 17a. This behavior is unique to the slat count 2 ribbons; for ribbons with slat counts 4 and 8, erroneous attachment below the point where errors become common rarely seems to result in stalled growth and instead typically results in uncontrolled growth as depicted in Figure 17b. This discrepancy between slat count 2 ribbons and those using slat counts of 4 or 8 can be explained by considering the point at which errors become common. For slat count 2, errors become common below $G_{mc}/G_{se} = 9$. At this ratio, slats still need several bound glues to attach stably and since growth is uncontrolled it's likely that all sites in which slats could attach stably quickly fill up. Compare this to the ribbons with slat counts 4 and 8 where errors are only common when G_{mc}/G_{se} is relatively small. Since only a few matching glues are required for stable attachment at these values of G_{mc}/G_{se}, it's much more likely that even during uncontrolled growth, there will be numerous sites in which slats can attach stably.

7.2 Details regarding ribbon systems with fixed total slat concentrations

Here we provide a few additional details regarding what we observed in the simulations of ribbons with varying amount of slat reuse in Section 4.4. Interestingly, the error rates for these



(a) A typical assembly from the simulation of our ribbons with slat count 2 at $G_{mc}/G_{se} = 7$. Here, errors have accumulated and filled in most sites surrounding the assembly which could admit stable attachments.

(b) A typical section from the growing edge of our simulated ribbons with slat count 8 at $G_{mc}/G_{se} = 2.5$. Here errors are plentiful and growth is uncontrolled.

Figure 17 Example assemblies from simulations of two different ribbon systems with cooperativity 16.

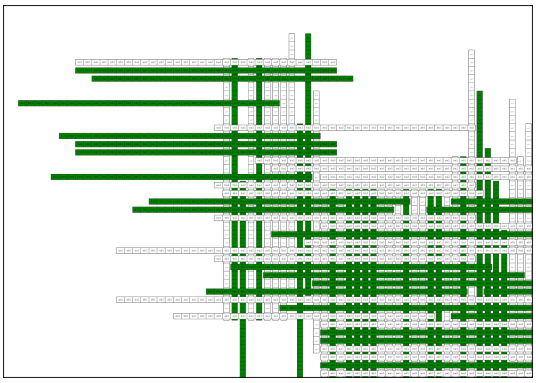


Figure 18 The growth front of a typical ribbon using slat count 2 with total slat concentration fixed at 480nM.

ribbons didn't seem to change significantly with the adjusted concentrations. Conversely, as should be expected, growth rates increased as slat counts decreased. These results agree well with the observations from [28] and suggest that by reducing slat counts further should be a viable approach to designing slat systems which grow more quickly. Suprisingly, unlike with the ribbons keeping individual slat counts fixed, growth of the ribbons using slat count 2 did not drop significantly when G_{mc}/G_{se} dropped below about 9. With individual slat concentrations fixed, the slat count 2 ribbons, reached a point where no further stable tile attachments were possible, however when total slat concentrations were fixed across ribbons, the increase to the individual slat concentrations allowed continued growth despite erroneous attachments being dominant. Figure 18 illustrates the growth front of such a ribbon when total slat concentrations were fixed which can be contrasted with Figure 17a for fixed individual slat counts.

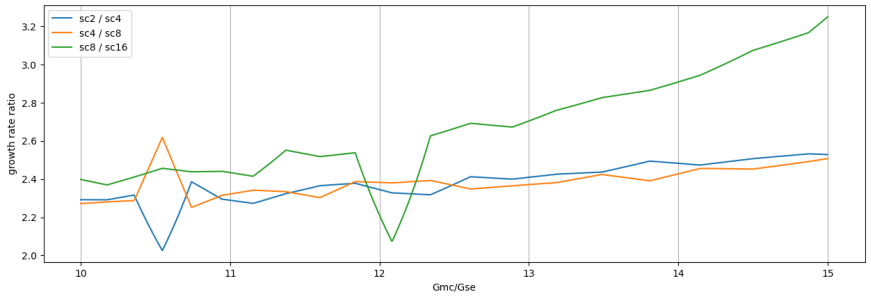


Figure 19 Ratios between growth rates for ribbons with slat counts differing by a factor of 2. Notice that this ratio never drops below 2 in the illustrated range, representing ideal conditions for error-free growth.

7.3 The Sierpinski triangle in tiles and slats

An example of a portion of the Sierpinski triangle self-assembled from aTAM tiles is shown in Figure 20, and (one layer of) our implementation with slats using cooperativity 4 is shown in Figure 22. This construction uses a gadget for each tile type from Figure 20, which can be seen in Figure 21.

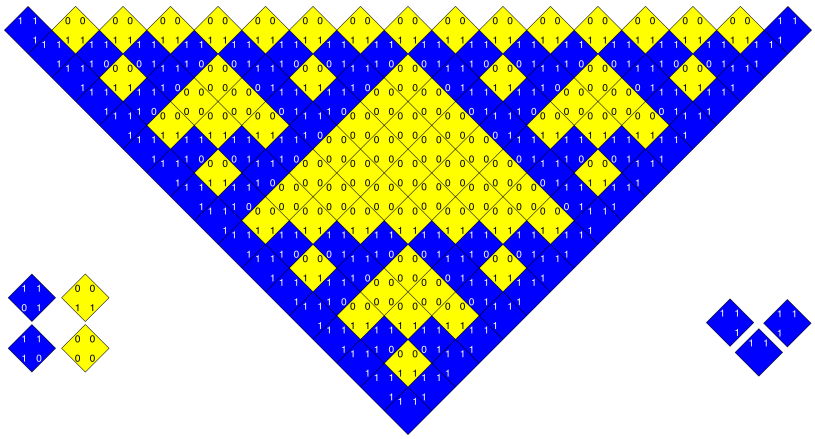


Figure 20 The Sierpinski triangle pattern. (left) The four "logic" tiles which have their inputs on the bottom and outputs on the top. The output bits are the logical "exclusive or" (`xor`) of the input bits, (middle) A portion of the infinite assembly, (right) The boundary tiles.

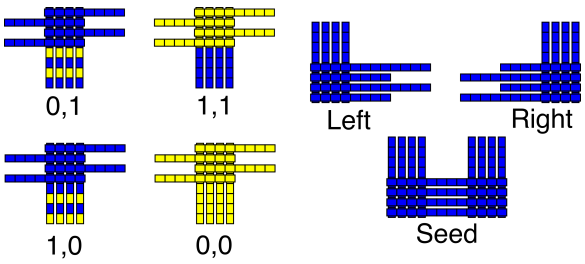


Figure 21 The gadgets of the Sierpinski triangle construction with slats using cooperativity 4 analogous to the tiles of the aTAM construction in Figure 20.

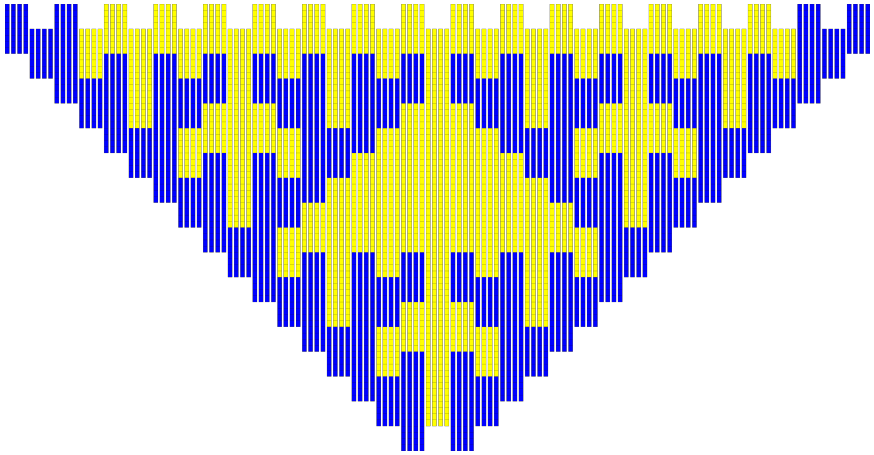


Figure 22 The Sierpinski triangle pattern made from a system of slats with cooperativity 4 (i.e. total length 8 each except for some boundary slats which are length 18). The gadgets of the construction are shown in Figure 21. Only the layer of vertical slats is shown, for clarity.

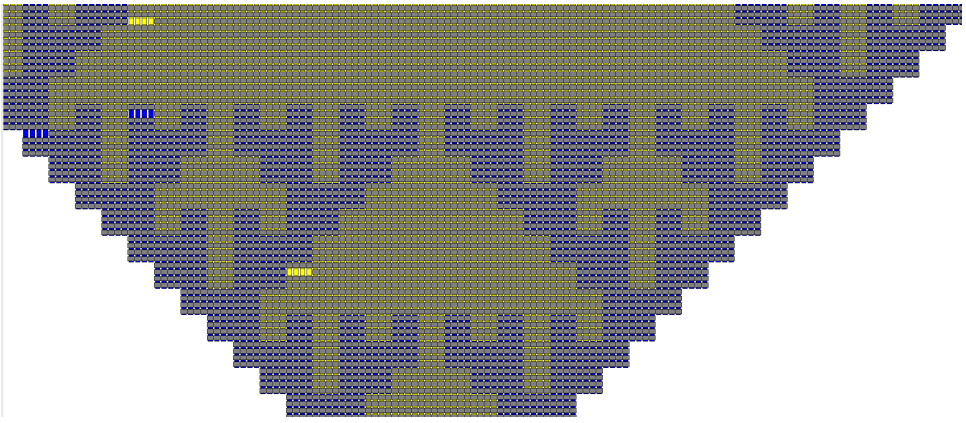


Figure 23 Growth errors during growth of a cooperativity 4 Sierpinski triangle using a slat count of 4 at $G_{mc}/G_{se} = 2.5$. Notice that some erroneous attachments are visible, leaving gaps where some horizontal slats should be, yet growth of the Sierpinski triangle pattern continues unaffected.

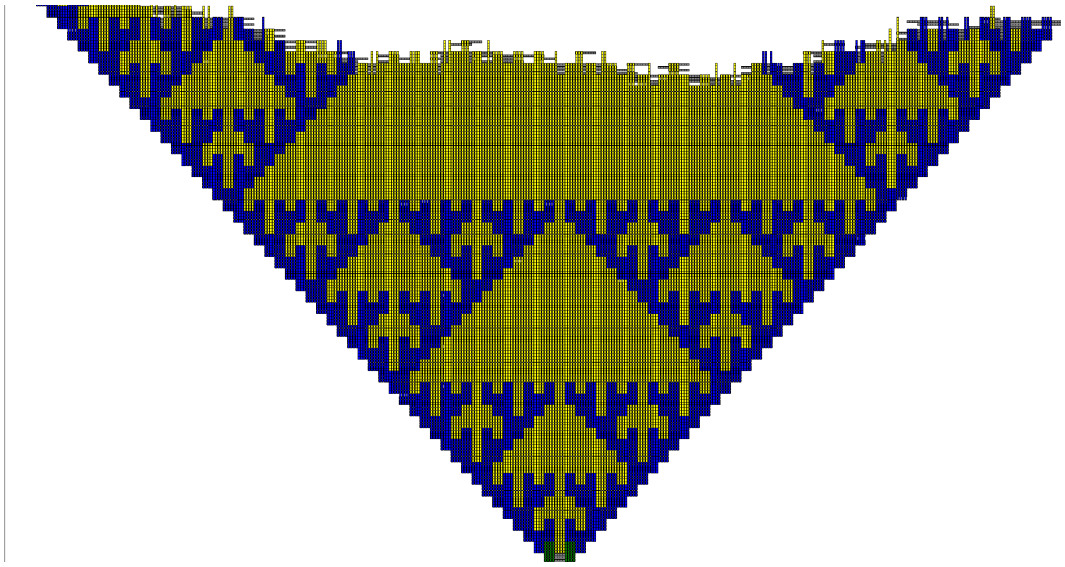


Figure 24 A zoomed out view of the complete assembly formed by the same simulation as illustrated in Figure 23. Here, horizontal slats are hidden so that the Sierpinski triangle pattern of the verical slats is more visible. This system was grown at $G_{mc}/G_{se} = 2.5$ where the probability of growth errors was distinctly non-zero. Still, the Sierpinski triangle pattern grows flawlessly.

Thermodynamically Driven Signal Amplification

Joshua Petrack
University of California–Davis, CA, USA

David Soloveichik
University of Texas at Austin, TX, USA

David Doty
University of California–Davis, CA, USA

Abstract

The field of chemical computation attempts to model computational behavior that arises when molecules, typically nucleic acids, are mixed together. By modeling this physical phenomenon at different levels of specificity, different operative computational behavior is observed. Thermodynamic binding networks (TBNs) is a highly abstracted model that focuses on which molecules are bound to each other in a "thermodynamically stable" sense. Stability is measured based only on how many bonds are formed and how many total complexes are in a configuration, without focusing on how molecules are binding or how they became bound. By defocusing on kinetic processes, TBNs attempt to naturally model the long-term behavior of a mixture (i.e., its thermodynamic equilibrium).

We study the problem of *signal amplification*: detecting a small quantity of some molecule and amplifying its signal to something more easily detectable. This problem has natural applications such as disease diagnosis. By focusing on thermodynamically favored outcomes, we seek to design chemical systems that perform the task of signal amplification robustly without relying on kinetic pathways that can be error prone and require highly controlled conditions (e.g., PCR amplification).

It might appear that a small change in concentrations can result in only small changes to the thermodynamic equilibrium of a molecular system. However, we show that it is possible to design a TBN that can "exponentially amplify" a signal represented by a single copy of a monomer called the *analyte*: this TBN has exactly one stable state before adding the analyte and exactly one stable state afterward, and those two states "look very different" from each other. In particular, their difference is exponential in the number of types of molecules and their sizes. The system can be programmed to any desired level of resilience to false positives and false negatives. To prove these results, we introduce new concepts to the TBN model, particularly the notions of a TBN's entropy gap to describe how unlikely it is to be observed in an undesirable state, and feed-forward TBNs that have a strong upper bound on the number of polymers in a stable configuration.

We also show a corresponding negative result: a *doubly* exponential upper bound, meaning that there is no TBN that can amplify a signal by an amount more than doubly exponential in the number and sizes of different molecules that comprise it. We leave as an open question to close this gap by either proving an exponential upper bound, or giving a construction with a doubly-exponential difference between the stable configurations before and after the analyte is added.

Our work informs the fundamental question of how a thermodynamic equilibrium can change as a result of a small change to the system (adding a single molecule copy). While exponential amplification is traditionally viewed as inherently a non-equilibrium phenomenon, we find that in a strong sense exponential amplification can occur at thermodynamic equilibrium as well – where the "effect" (e.g., fluorescence) is exponential in types and complexity of the chemical components.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Thermodynamic binding networks, signal amplification, integer programming

Digital Object Identifier 10.4230/LIPIcs.DNA.29.8

Related Version *Full Version*: `https://arxiv.org/abs/2307.01550`

Funding *Joshua Petrack*: NSF grants 1900931 and 1844976.
David Soloveichik: NSF grant 1901025, Sloan Foundation Research Fellowship.
David Doty: NSF grants 2211793, 1900931, and 1844976.

1 Introduction

Detecting a small amount of some chemical signal, or analyte, is a fundamental problem in the field of chemical computation. The current state-of-the-art in nucleic acid signal amplification is the polymerase chain reaction (PCR)[9]. By using a thermal cycler, PCR repeatedly doubles the amount of the DNA strand that is present. One downside is the need for a PCR machine, which is expensive and whose operation can be time-consuming. The advantages of PCR are that it can reliably detect even a single copy of the analyte if enough doubling steps are taken, and it is fairly (though not perfectly) robust to incorrect results. Recent work in DNA nanotechnology achieves "signal amplification" through other kinetic processes involving pure (enzyme-free) DNA systems, such as hybridization chain reaction (HCR) [4], classification models implemented with DNA [7], hairpin assembly cascades [10], and "crisscross" DNA assembly [8].

Although highly efficacious, PCR and these other techniques essentially rely on *kinetic* control of chemical events, and the thermodynamic equilibria of these systems are not consistent with their desired output. Can we design a system so that, if the analyte is present, the thermodynamically most stable state of the system looks one way, and if the analyte is absent, the thermodynamically most stable state looks "very different" (e.g., many fluorophores have been separated from quenchers)? Besides answering a fundamental chemistry question, such a system is potentially more robust to false positives and negatives. It also can be simpler and cheaper to operate: for many systems, heating up the system and cooling it down slowly reaches the system's thermodynamic equilibrium.

We tackle this problem of signal detection in the formal model of Thermodynamic Binding Networks (TBNs) [5, 3]. The TBN model of chemical computation ignores kinetic and geometric constraints in favor of focusing purely on configurations describing which molecules are bound to which other molecules. A TBN yields a set of stable configurations, the ways in which monomers (representing individual molecules, typically strands of DNA) are likely to be bound together in thermodynamic equilibrium. A TBN performs the task of signal amplification if its stable configurations, and thus the states in which it is likely to be observed at equilibrium, change dramatically in response to adding a single monomer. TBNs capture a notion of what signal amplification can look like for purely thermodynamic chemical systems, without access to a process like PCR that repeatedly changes the conditions of what is thermodynamically favorable.

This paper asks the question: if we add a single molecule to a pre-made solution, how much can that change the solution's thermodynamic equilibrium? To make the question quantitative, we define a notion of distance between thermodynamic equilibria, and we consider scaling with respect to meaningful complexity parameters. First, we require an upper limit on the size of molecules in the solution and the analyte, as adding a single very large molecule can trivially affect the entire solution. Large molecules are also expensive to synthesize, and for natural signal detection the structure of the analyte is not under our control. Second, we require an upper limit on how many different types of molecules are in the solution, as it is expensive to synthesize new molecular species (though synthesizing many copies is more straightforward).

Our main result is the existence of a family of TBNs that amplify signal exponentially. In these TBNs, there are exponentially many free "reporter" monomers compared to the number of types of monomers and size of monomers. In the absence of the analyte, this TBN has a unique stable configuration in which all reporter monomers are bound. When a single copy of the analyte is added, the resulting TBN has a unique stable configuration in

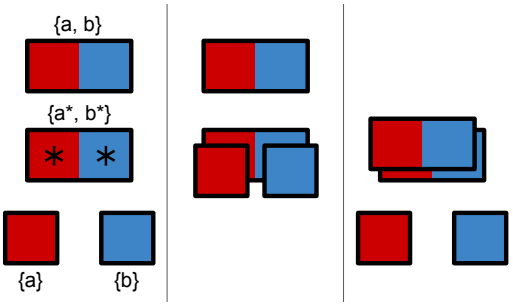


Figure 1 A simple thermodynamic binding network T with four monomers. Site types are differentiated by color. Bonds are shown by juxtaposing monomers so that unstarred sites cover starred sites. Left: the all-singleton configuration melt(T) with four polymers. Monomers are labeled by their formal identities for reference. Middle: a configuration with two polymers. As all starred sites are covered, the configuration is saturated. Right: a configuration with three polymers. As this has the most possible polymers for a saturated configuration, it is stable.

which all reporter monomers are unbound. These TBNs are parameterized by two values: the first is the amplification factor, determining how many total reporter molecules are freed. The second is a value we call the system's "entropy gap", which determines how thermodynamically unfavorable a configuration of the system would need to be in order for reporters to be spuriously unbound in the absence of the analyte (false positive) or spuriously bound in its presence (false negative).

We also show a corresponding doubly exponential upper bound on the signal amplification problem in TBNs: that given any TBN, adding a single monomer can cause at most a doubly exponential change in its stable configurations. We leave as an open question to close this gap: either proving an exponential upper bound, or giving a TBN with a doubly-exponential amplification factor.

Our work can be compared to prior work on signal amplification that exhibits kinetic barriers. For example, in reference [8], a detected analyte serves as a seed initiating self-assembly of an arbitrarily long linear polymer. In the absence of the analyte, an unlikely kinetic pathway is required for spurious nucleation of the polymer to occur. However, in that system, false positive configurations are still thermodynamically favorable; if a critical nucleus is able to overcome the kinetic barrier and assemble, then growth of the infinite polymer is equally favorable as from the analyte. In contrast, in our system, there are *no* kinetic paths, however unlikely, that lead to an undesired yet thermodynamically favored configuration.

2 Definitions

2.1 General TBN Definitions

A *site type* is a formal symbol such as a, and has a *complementary* type, denoted a^*, with the interpretation that a binds to a^* (e.g., they could represent complementary DNA sequences). We also refer to site types as domain types, and sites as domains. We call a site type such as a an *unstarred* site type, and a^* a *starred* site type. A *monomer type* is a multiset of site types (e.g., a DNA strand consisting of several binding domains); for example monomer type $\vec{m} = \{a, a, a, b, c^*\}$ has three copies of site a, one of site b, and one of site c^*. A *TBN* [5, 2] is a multiset of monomer types. We call an instance of a monomer type a *monomer* and an instance of a site type a *site*.

We take a convention that, unless otherwise specified, TBNs are *star-limiting*: for each site type, there are always at least as many sites of the unstarred type as the starred type among all monomers. Given a TBN, this can always be enforced by renaming site types to swap unstarred and starred types, which simplifies many of the definitions below.

A *configuration* of a TBN is a partition of its monomers into submultisets called *polymers*. We say that a site type (or a site) on a polymer is *uncovered* if, among the monomers in that polymer, there are more copies of the starred version of that site type than the unstarred version (otherwise *covered*). A polymer is *self-saturated* if it has no uncovered site types. A configuration is *saturated* if all its polymers are self-saturated. A configuration α of a TBN T is *stable* if it is saturated, and no saturated configuration of T has more polymers than α. Figure 1 shows an example TBN.

An equivalent characterization of stability is in terms of merges rather than polymer counts. We say that a *merge* is the process of taking two polymers in a configuration and making a new configuration by joining them into one polymer; likewise a *split* is the process of taking one polymer in a configuration and making a new configuration by splitting it into two polymers. Maximizing the number of polymers in a saturated configuration is equivalent to minimizing the number of merges of two polymers necessary to reach a saturated configuration. To this end, some additional notation:

▶ **Definition 2.1.** *The* distance to stability *of a saturated configuration σ is the number of (splits minus merges) necessary to get from σ to a stable configuration.*

Note that this number will be the same for any path of splits and merges, as all stable configurations have the same number of polymers.

Equivalently, distance to stability is the number of polymers in a stable configuration minus the number of polymers in σ. We only consider this value for saturated configurations to ensure it is positive and because we may interpret it as a measure of how unlikely we are to observe the network in a given state under the assumption that enthalpy matters infinitely more than entropy.

The following definitions are not restricted to saturated configurations.

▶ **Definition 2.2.** *Given a TBN T, we say that the all-melted configuration, denoted* $\mathrm{melt}(T)$*, is the configuration in which all monomers are separate.*

▶ **Definition 2.3.** *Given a configuration α in a TBN T, its* merginess $m(\alpha)$ *is the number of merges required to get from* $\mathrm{melt}(T)$ *to α (or equivalently, the number of monomers in T minus the number of polymers in α).*

▶ **Definition 2.4.** *Given a configuration α in a TBN T, its* starriness $s(\alpha)$ *is the number of polymers in α which contain at least one uncovered starred site.*

We observe that α is saturated if and only if $s(\alpha) = 0$.

▶ **Definition 2.5.** *Given configurations α and β in a TBN T, we say $\alpha \preceq \beta$ (equivalently, $\beta \succeq \alpha$) if it is possible to reach β from α solely by splitting polymers zero or more times.*

We read $\alpha \preceq \beta$ as "α splits to β". Observe that if $\alpha \succeq \beta$, then we can reach β from α in exactly $m(\beta) - m(\alpha)$ merges. In general, we may order the merges required to go from one configuration to another in whatever way allows the easiest analysis.

2.2 Comparing TBNs

We need some notion of how "different" two TBNs are, so that we can quantify how much a TBN changes after adding a single monomer.

▶ **Definition 2.6** (distance between configurations). *Let α and β be two configurations of a TBN, or of two TBNs using the same monomer types. We say that the* distance $d(\alpha, \beta)$ *between them is the L^1 distance between the vectors of their polymer counts. That is, it is the sum over all types of polymers of the difference between how many copies of that polymer are in α and in β.*

▶ **Definition 2.7** (distance between TBNs). *Given TBNs T and T', let $\mathcal{C}$ and $\mathcal{C}'$ be their stable configurations. Define the distance between T and T' as*

$$d(T, T') = \min_{\alpha \in \mathcal{C}, \alpha' \in \mathcal{C}'} d(\alpha, \alpha'). \tag{1}$$

Note that this distance is not a metric.[1] Rather, it is a way to capture how easily we can distinguish between two TBNs; even the closest stable configurations of T and T' have distance $d(T, T')$, so we should be able to distinguish any stable configuration of one of them from all stable configurations of the other by that amount.

Note that this condition does not directly imply a stronger "experimentally verifiable" notion of distance, namely that there is some "reporter" monomer which is always bound in one TBN and always free in the other. However, the system we exhibit in this paper does also satisfy this stronger condition. We focus on the distance given here, as it is more theoretically general and our upper bound result in Section 4 apply to it.

We also need a notion of how likely we are to observe a configuration of a TBN that is not stable, in order to have a notion of the system being robust to random noise. If a TBN has one stable configuration but many other configurations that are nearly stable, we would expect to observe it in those configurations frequently, meaning that in practice we may not be able to discern what the stable configuration is as easily.

We work under the assumption that enthalpy matters infinitely more than entropy, so that we may assume that only saturated configurations need to be considered. This assumption is typical for the TBN model, and can be accomplished practically by designing binding sites to be sufficiently strong. Under this paradigm, a configuration's distance to stability is a measure of how unlikely we are to observe it. This motivates the following definition:

▶ **Definition 2.8** (entropy gap). *Given a TBN T, we say that it has an* entropy gap *of k if, for any saturated configuration α of T, one of the following is true:*

1. *α is stable.*
2. *There exists some stable configuration β such that $\alpha \preceq \beta$.*
3. *α has distance to stability at least k.*

Note that by this definition, all TBNs trivially have an entropy gap of one. Note as well that stable configurations are technically also included in the second condition by choosing $\beta = \alpha$, but we list them separately for emphasis.

The second condition is necessary in this definition because any TBN necessarily has some configurations that have distance to stability one, simply by taking a stable configuration and arbitrarily merging two polymers together. These configurations are unavoidable but

[1] In particular, it fails to satisfy the triangle inequality, since T could have a stable configuration close to one of T', so $d(T, T') = 1$, and T' could have a different stable configuration close to one of T'', so $d(T', T'') = 1$, but T and T'' could have no close stable configurations, so $d(T, T'') > 2$.

are not likely to be problematic in a practical implementation, because a polymer in such a configuration should be able to naturally split itself without needing to interact with anything else – these configurations will never be local energy minima. Reference [2] discusses self-stabilizing TBNs in which *all* saturated configurations have this property, equivalent to an entropy gap of ∞.

2.3 Feed-Forward TBNs

▶ **Definition 2.9.** *We say that a configuration α of a TBN is* feed-forward *if there is an ordering of its polymers such that for each domain type, all polymers with an excess of unstarred instances of that domain type occur before all polymers with an excess of starred instances of that domain type.*

We say that a TBN T is feed-forward *if there is an ordering of its monomer types with this same property – that is, T is feed-forward if* $\text{melt}(T)$ *is feed-forward.*

For example, the TBN $\{(ab), (a^*c), (b^*c^*)\}$ is feed-forward with this ordering of monomers because the a, b and c come strictly before the a^*, b^* and c^* respectively. Note that not all configurations of a feed-forward TBN are necessarily feed-forward; for instance, merging the first and third monomers in this TBN gives a non-feed-forward configuration.

An equivalent characterization can be obtained by defining a directed graph on the polymers of a configuration α and drawing an edge between any two polymers that can bind to each other, from the polymer with an excess unstarred binding site to the polymer with a matching excess starred binding site (or both directions if both are possible). The configuration α is feed-forward if and only if this graph is acyclic, and the ordering of polymers can be obtained by taking a topological ordering of its vertices.

The main benefit of considering feed-forward TBNs is that we can establish a strong lower bound on the merginess of stable configurations. If *any* TBN T has n monomers that have starred sites, it will always take at least $\frac{n}{2}$ merges to cover all those sites, because each monomer must be involved in at least one merge and any merge can at most bring a pair of them together. For instance, the non-feed-forward TBN $\{\{a, b^*\}, \{a^*, b\}\}$ can be stabilized with a single merge. In feed-forward TBNs, this bound is even stronger, as there is no way to "make progress" on covering the starred sites of two different monomers at the same time.

▶ **Lemma 2.10.** *If a configuration α is feed-forward, then any saturated configuration σ such that $\alpha \succeq \sigma$ satisfies $m(\sigma) - m(\alpha) \geq s(\alpha)$. That is, reaching σ from α requires at least $s(\alpha)$ additional merges.*

Intuitively, in a feed-forward configuration, the best we can possibly do is to cover all of the starred sites on one polymer at a time. We can never do better than this with a merge like merging $\{a, b^*\}$ and $\{a^*, b\}$ that would let two polymers cover all of each others' starred sites.

Proof. Given a feed-forward configuration α, let L be the ordered list of polymers from α being feed-forward. Partition L into separate lists (keeping the ordering from L) based on which polymers are merged together in σ. That is, for each fully merged polymer $\mathbf{P} \in \sigma$ create a list $L_{\mathbf{P}}$ of the polymers from α that are merged to form $\mathbf{P}$, and order this list based on the ordering from L. We can order the merges to reach σ from α as follows: repeatedly (arbitrarily) pick a polymer $\mathbf{P}$ from σ and merge all of the polymers in $L_{\mathbf{P}}$ together in order (merge the first two polymers in $L_{\mathbf{P}}$, then merge the third with the resulting polymer, and so on).

This sequence of merges gives us a sequence of configurations $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_\ell = \sigma$. We observe that for $1 \leq i \leq \ell - 1$, we have $s(\alpha_i) - s(\alpha_{i+1}) \leq 1$. That is, each merge can lower the starriness by at most one. We know this because each merge is merging a polymer $\mathbf{Q} \in \alpha$ with one or more other already-merged polymers from α that all come before $\mathbf{Q}$ in L. This means $\mathbf{Q}$ cannot cover any starred sites on any monomers it is merging with. The only way for the starriness of a configuration to decrease by more than 1 in a single merge is for the two merging polymers to cover all of each others' starred sites, so it follows that each merge in this sequence lowers starriness by at most 1. From this it follows that we need at least $s(\alpha)$ merges to get to σ, because $s(\sigma) = 0$. ◀

Letting $\alpha = \text{melt}(T)$ (note $m(\alpha) = 0$) gives the following corollary.

▶ **Corollary 2.11.** *Any saturated configuration σ of a feed-forward TBN T satisfies $m(\sigma) \geq s(\text{melt}(T))$.*

Because stable configurations are saturated configurations with the minimum possible merginess, this bound gives the following corollary.

▶ **Corollary 2.12.** *If a saturated configuration σ of a feed-forward TBN T satisfies $m(\sigma) = s(\text{melt}(T))$, then σ is stable.*

3 Signal Amplification TBN

3.1 Amplification Process

In this section, we prove our main theorem. This theorem shows the existence of a TBN parameterized by two values n (the amplification factor) and k (the entropy gap). Intuitively, this TBN amplifies the signal of a single monomer by a factor of 2^n, with any configurations that give "incorrect" readings having $\Omega(k)$ distance to stability. Our proof will be constructive.

▶ **Theorem 3.1.** *For any integers $n \geq 1, k \geq 2$, there exists a TBN $T = T_{n,k}$ and monomer $\mathbf{a}$ (the analyte) such that if $T^{\text{a}} = T^{\text{a}}_{n,k}$ is the TBN obtained by adding one copy of $\mathbf{a}$ to $T_{n,k}$, then*

1. *T and T^{a} each have exactly one stable configuration, denoted $\sigma_{n,k}$ and $\sigma^{\text{a}}_{n,k}$ respectively, with $d(\sigma_{n,k}, \sigma^{\text{a}}_{n,k}) \geq 2^n$. In particular, there are k monomer types with 2^{n-1} copies each, with all of these monomers bound in $\sigma_{n,k}$ and unbound in $\sigma^{\text{a}}_{n,k}$.*
2. *T and T^{a} each have an entropy gap of $\lfloor \frac{k}{2} \rfloor - 1$.*
3. *T and T^{a} each use $\mathcal{O}(nk)$ total monomer types, $\mathcal{O}(nk^2)$ domain types, and $\mathcal{O}(k^2)$ domains per monomer.*

The first condition implies that $T_{n,k}$ can detect a single copy of $\mathbf{a}$ with programmable exponential strength - there is only one stable configuration either with or without $\mathbf{a}$, and they can be distinguished by an exponential number of distinct polymers. Note that this is even stronger than saying that $d(T_{n,k}, T^{\text{a}}_{n,k}) \geq 2^n$, as that statement would allow each TBN to have multiple stable configurations. The second condition implies that the system has a programmable resilience to having incorrect output, because configurations other than the unique stable ones in each case are "programmably" unstable (based on k), and thus programmably unlikely to be observed. Note that throughout this paper we will use $\frac{k}{2}$ instead of $\lfloor \frac{k}{2} \rfloor$ for simplicity, as we are concerned mainly with asymptotic behavior. The third condition establishes that the system doesn't "cheat" - it doesn't obtain this amplification by either having an extremely large number of distinct monomers, or by having any single large monomers.

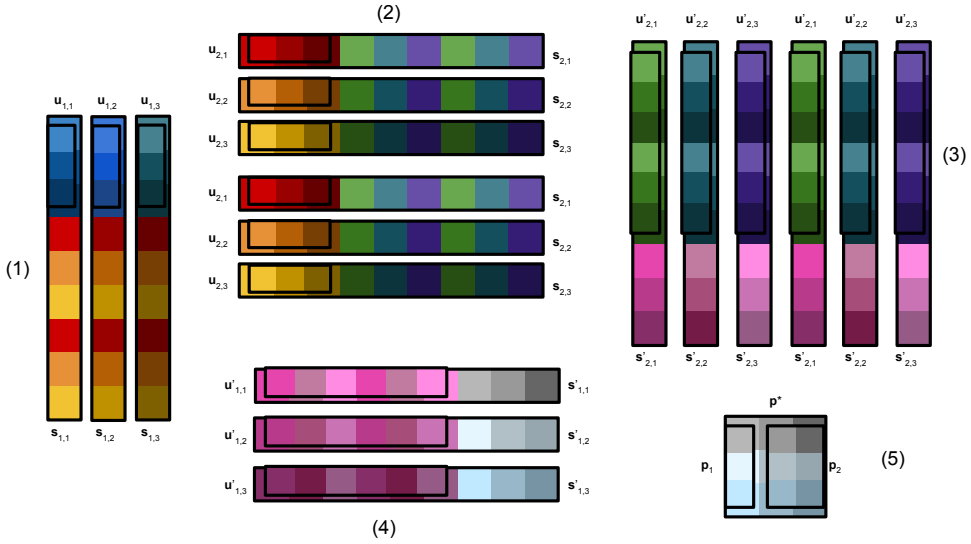


Figure 2 The unique stable configuration $\sigma_{2,3}$ of $T_{2,3}$, with 19 polymers. All starred sites are visually "covered" by unstarred sites on another monomer. The parts of the diagram are numbered by the order that the signal from the analyte will cascade through them. Parts (1) and (2) form the "first half", where the signal is doubled at each step. Parts (3) and (4) form the "second half", where the signal converges so that it can get an "entropic payoff" from part (5).

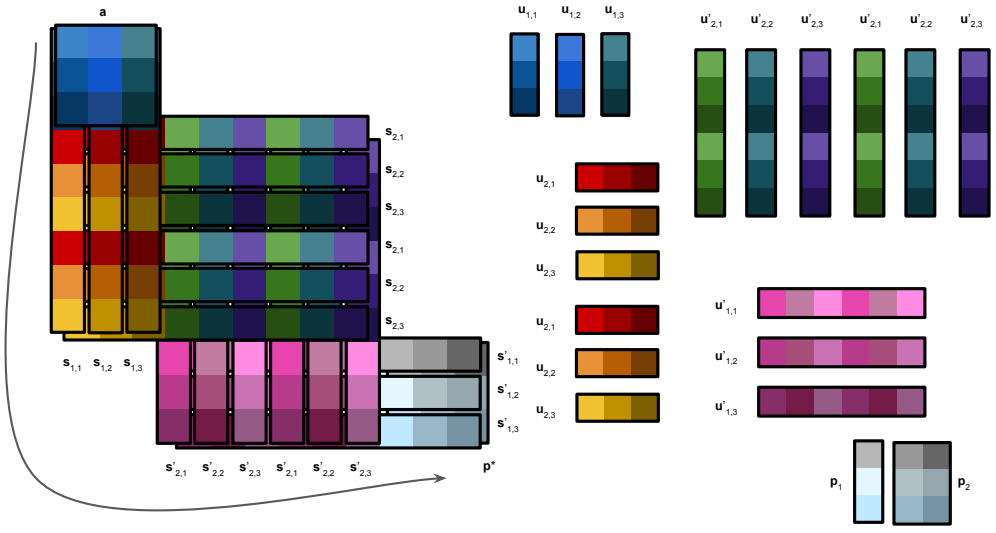


Figure 3 The unique stable configuration $\sigma^{\mathrm{a}}_{2,3}$ of $T^{\mathrm{a}}_{2,3}$. The arrow shows the conceptual order in which the analyte's signal has been propagated, with $\mathbf{a}$ covering all $\mathbf{s}_{1,j}$, which cover all $\mathbf{s}_{2,j}$, which cover all $\mathbf{s}'_{2,j}$, which cover all $\mathbf{s}'_{1,j}$, which finally cover $\mathbf{p}^*$. This configuration has 21 polymers, 2 more than $\sigma_{2,3}$: conceptually, one of these is from adding the analyte and the other is from the analyte's signal cascading through the layers to release P_1 and P_2 at the cost of one merge. As they have no polymers in common, $d(\sigma_{2,3}, \sigma^{\mathrm{a}}_{2,3}) = 19 + 21 = 40$.

The entire TBN $T_{n,k}$ is depicted in Figures 2 and 3 with $n = 2$ and $k = 3$. The former shows the unique stable configuration before adding the analyte, and the latter shows the unique stable configuration after adding the analyte. For comparison, Figure 7 depicting the pre-analyte configuration with $n = 3$ and $k = 2$ is shown in the appendix.

We will start by constructing the first half of $T_{n,k}$ and describing "how it works". The monomers in this first half are the driving force that allows $T_{n,k}$ and $T^{\mathrm{a}}_{n,k}$ to have exponentially different stable configurations.

The first half of $T_{n,k}$ has monomer types $\mathbf{u}_{i,j}$ and $\mathbf{s}_{i,j}$ (named as those with only unstarred sites, and those with both starred and unstarred sites) for $1 \leq i \leq n, 1 \leq j \leq k$. It has domain types denoted as triples (i, j, ℓ) for $1 \leq i \leq n+1, 1 \leq j, \ell \leq k$. Each $\mathbf{u}_{i,j}$ monomer has k different unstarred domains, one of each (i, j, ℓ) for each $1 \leq \ell \leq k$. Each $\mathbf{s}_{i,j}$ monomer has a starred copy of each domain in $\mathbf{u}_{i,j}$, and additionally has two copies of each unstarred domain $(i + 1, \ell, j)$ for each $1 \leq \ell \leq k$ (note that here the second domain type parameter varies instead of the third). For each $\mathbf{u}_{i,j}$ and $\mathbf{s}_{i,j}$ monomer, there are 2^{i-1} copies. We can conceptually break these monomers into n "layers", each consisting of all monomers with the same value for their first parameter. The analyte we wish to detect, $\mathbf{a}$, is a monomer that has one copy of each unstarred domain $(1, j, \ell)$, $1 \leq j, \ell \leq k$.

Conceptually, when the analyte is absent, the most efficient way for all starred sites on each $\mathbf{s}_{i,j}$ to be covered is by the unstarred sites on a corresponding $\mathbf{u}_{i,j}$, as seen in Figure 2. Although the TBN model is purely thermodynamic, we can conceptualize that when the analyte is added, its signal can propagate "kinetically" through each layer. In the first layer, it can "displace" the k different $\mathbf{u}_{1,j}$ monomers and bind to all of the $\mathbf{s}_{1,j}$ monomers. In doing so, it brings together all the unstarred sites on all of the $\mathbf{s}_{1,j}$ monomers. Having been brought together, these sites "look like" two copies of the analyte, but with the domains from layer 2 instead of layer 1. Thus, this polymer is then able to displace *two* copies of each $\mathbf{u}_{2,j}$ from their corresponding $\mathbf{s}_{2,j}$ monomers, thus bringing all of the $\mathbf{s}_{2,j}$ together. This in turn now looks like four copies of the analyte for the domains in the third layer, and so on. Each layer allows this polymer to assimilate exponentially more $\mathbf{s}_{i,j}$, thus freeing exponentially many $\mathbf{u}_{i,j}$. Each of these displacement steps involves an equal number of splits and merges.

3.2 Convergence Process

So far, the TBN described has exactly one stable configuration before adding the analyte, and it performs the task of amplifying signal by having the potential to change its state exponentially when the analyte is added. However, there is also a stable configuration after adding the analyte in which nothing else changes, and many others in which only a small amount of change occurs. We must guaranteed that the analyte's signal "propagates" through all of the layers.

To design the system to meet this requirement, we observe that all exponentially many monomers that have been brought together must contribute to some singular change in the system that gains some entropy, to spur the signal into propagating. The typical way to accomplish this in a TBN is by having monomers that have been brought together displace a larger number of monomers from some complex at the cost of a smaller number of merges. Because the pre-analyte TBN has an entropy gap of $k - 1$ in this design so far, we can afford to give the TBN with the analyte an "entropic payoff" of $\frac{k}{2}$. When the analyte is absent, this payoff is weak enough that there will still be an entropy gap of $\frac{k}{2} - 1$; when the analyte is present, the existence of this payoff will force the signal to fully propagate, and will give the TBN with the analyte an entropy gap of $\frac{k}{2} - 1$ by making it so that any configurations in which this payoff is not achieved are also far away from stable.

Another challenge is that we cannot simply detect all our exponentially many conjoined monomers by binding them all to a single exponentially large monomer, because we need to bound the size of the largest monomer in the system. Our conceptual strategy for overcoming this is as follows: the signal will converge in much the same way as it was amplified. In the amplification step, one set of domains coming together in one layer was enough to cause two of them to come together in the next layer. In this convergence step, two sets of domains in one layer will have to converge together to activate one set in the next layer. This convergence ends in bringing together a set of binding sites that is of the same size as the analyte, which can then directly displace some monomers to gain $\frac{k}{2}$ total polymers.

We now fully define $T_{n,k}$. We start with the already described $\mathbf{u}_{i,j}$ and $\mathbf{s}_{i,j}$. To these, we first add monomer types $\mathbf{u}'_{i,j}$ and $\mathbf{s}'_{i,j}$ for $1 \leq i \leq n, 1 \leq j \leq k$. These monomers are the "converging" equivalents of $\mathbf{u}_{i,j}$ and $\mathbf{s}_{i,j}$. Conceptually, they will activate in the reverse order: two copies of each $\mathbf{s}'_{i,j}$ for $1 \leq j \leq k$, when brought together, will be able to bring together one copy of each $\mathbf{s}'_{i-1,j}$ for $1 \leq j \leq k$.

Each $\mathbf{u}'_{i,j}$ monomer has $2k$ unstarred domains: two copies each of domains $(i+1, j, \ell)'$ for each $1 \leq \ell \leq k$. Each $\mathbf{s}'_{i,j}$ has a starred copy of each of the $2k$ domains in $\mathbf{u}'_{i,j}$ and additionally has one unstarred domain $(i, \ell, j)'$ for $1 \leq \ell \leq k$ (note again here that the second domain type parameter varies instead of the third). One exception is the monomers $\mathbf{u}'_{n,j}$ and $\mathbf{s}'_{n,j}$ (the first ones to activate) which use domains $(n+1, j, \ell)$ and $(n+1, \ell, j)$ instead of $(n+1, j, \ell)'$ and $(n+1, \ell, j)'$ respectively so that they can interact with $\mathbf{s}_{n,j}$ monomers that have been brought together. Each monomer $\mathbf{u}'_{i,j}$ and $\mathbf{s}'_{i,j}$ has 2^{i-1} copies.

Finally, we add "payoff" monomers that will yield an entropic gain of $\frac{k}{2}$ when the signal from the analyte has cascaded through every layer. This choice of $\frac{k}{2}$ is arbitrary – a similar design works for any integer between 1 and k. Choosing a higher value leads to a higher entropy gap after adding the analyte and a lower entropy gap before adding it, and vice versa choosing a lower value. For simplicity of definitions we will assume k is even (though figures are shown with $k = 3$, which shows how to generalize to odd k).

We add one monomer $\mathbf{p}^*$, which contains the k^2 sites $(1, \ell_1, \ell_2)'^*$ for $1 \leq \ell_1, \ell_2 \leq k$. Note that this monomer can be replaced with k monomers of size k (in which case $\mathbf{a}$ would be the only monomer with more than $3k$ domains), but doing so makes the proof more complex. The idea is that when all $\mathbf{s}'_{1,j}$ monomers are already together (as they can be "for free" when $\mathbf{a}$ is present), they can cover $\mathbf{p}^*$ in one merge; if they are apart, this requires k merges. In order to make this favorable to happen when they're already together but unfavorable when they're initially apart, we add another way to cover $\mathbf{p}^*$ that takes $\frac{k}{2}$ merges. This is accomplished via monomers $\mathbf{p}_j$ for $1 \leq j \leq \frac{k}{2}$. Each $\mathbf{p}_j$ contains the $2k$ sites $(1, 2j-1, \ell)$ and $(1, 2j, \ell)$ for $1 \leq \ell \leq k$. We can interpret this geometrically as $\mathbf{p}^*$ being a square, the $\mathbf{s}'_{1,j}$ covering it by rows, and the $\mathbf{p}_j$ covering it by two columns at a time. This completes the definition of $T_{n,k}$. Recall $T^{\mathrm{a}}_{n,k}$ is $T_{n,k}$ with one added copy of $\mathbf{a}$.

▶ **Lemma 3.2.** *$T_{n,k}$ has exactly one stable configuration $\sigma_{n,k}$.*

▶ **Corollary 3.3.** *$T_{n,k}$ has an entropy gap of $\frac{k}{2} - 1$.*

▶ **Lemma 3.4.** *$T^{\mathrm{a}}_{n,k}$ has exactly one stable configuration $\sigma^{\mathrm{a}}_{n,k}$, and $T^{\mathrm{a}}_{n,k}$ has an entropy gap of $\frac{k}{2} - 1$.*

Proofs of these results are left to the appendix.

These results together complete the proof of Theorem 3.1: each of the more than 2^n $\mathbf{u}$ and $\mathbf{u}'$ monomers (which serve as reporters) are bound in $\sigma_{n,k}$ and unbound in $\sigma^{\mathrm{a}}_{n,k}$, implying their distance is more than 2^n. The largest monomer is $\mathbf{a}$ with k^2 domains, and there are $(2n+1)k^2$ domain types and $4nk$ monomer types for the $\mathbf{s}_{i,j}$, $\mathbf{u}_{i,j}$, $\mathbf{s}'_{i,j}$, and $\mathbf{u}'_{i,j}$, plus $2 + \frac{k}{2}$ more for $\mathbf{a}$, $\mathbf{p}^*$, and $\mathbf{p}_j$.

3.3 Avoiding Large Polymer Formation

The TBN $T^{\mathrm{a}}_{n,k}$ will, in the process of amplifying the signal of the analyte, form a single polymer of exponential size. This isn't an issue in the theoretical TBN model, but it is a practical issue because there is no way to design these monomers so that this large polymer would form.[2]

This can be solved by adding "translator gadgets". These gadgets' job is to mediate between consecutive layers. Instead of monomers from one layer directly binding to monomers from the next layer, they can split apart these translator gadgets with half of the gadget going to each layer. In exchange, the TBN will no longer have exactly one stable configuration when the analyte is present, as in the TBN model, the use of these translator gates will be purely "optional".

We define a new TBN $\widetilde{T}_{n,k}$ (as well as $\widetilde{T}^{\mathrm{a}}_{n,k}$, which is obtained by adding the analyte $\mathbf{a}$). We start with the TBN $T_{n,k}$. To assist with the amplification step, we add monomer types $\mathbf{g}_i$ and $\mathbf{g}^*_i$ for each $2 \leq i \leq n$. Each $\mathbf{g}_i$ consists of one copy of each unstarred domain (i, j, ℓ) for each $1 \leq j, \ell \leq k$. Each $\mathbf{g}^*_i$ consists of the same domains but all starred. Each of these monomers has 2^{i-1} copies. The use of these gadgets can be seen in Figure 4.

To assist with the convergence step, we add monomer types $\mathbf{h}_i$ and $\mathbf{h}^*_i$ for each $2 \leq i \leq n+1$. Each $\mathbf{h}_i$ has *two* copies of each unstarred domain $(i, j, \ell)'$ for each $1 \leq j, \ell \leq k$. Each $\mathbf{h}^*_i$ has only one copy of each of the corresponding starred domains. There are 2^{i-1} copies of each $\mathbf{h}_i$ and 2^i copies of each $\mathbf{h}^*_i$. The use of these gadgets can be seen in Figure 5.

▶ **Theorem 3.5.** *Let $\widetilde{T} = \widetilde{T}_{n,k}$ and $\widetilde{T}^{\mathrm{a}} = \widetilde{T}^{\mathrm{a}}_{n,k}$ be as described. Then:*

1. *$\widetilde{T}$ has exactly one stable configuration $\widetilde{\sigma}_{n,k}$, and $d(\widetilde{T}, \widetilde{T}^{\mathrm{a}}) > 2^n$.*
2. *$\widetilde{T}$ has an entropy gap of $\frac{k}{2}$, and $\widetilde{T}^{\mathrm{a}}$ has the property that all of its configurations α that are within distance to stability $\frac{k}{2}$ satisfy $d(\widetilde{\sigma}_{n,k}, \alpha) > 2^n$.*
3. *$\widetilde{T} = \widetilde{T}_{n,k}$ uses $\mathcal{O}(nk)$ total monomer types, $\mathcal{O}(nk^2)$ domain types, and $\mathcal{O}(k^2)$ domains per monomer.*
4. *The unique stable configuration of $\widetilde{T}$ has $\mathcal{O}(k)$ monomers in its largest polymer. There is a stable configuration of $\widetilde{T}^{\mathrm{a}}$ sharing this property.*

Compared to Theorem 3.1, this theorem trades away the condition that both TBNs have only a single stable configuration in exchange for the post-analyte TBN having a configuration with $\mathcal{O}(k)$ monomers per polymer, whereas the previous construction has roughly $k \cdot 2^n$ monomers in a single polymer.

The second condition is somewhat complex. This complexity's necessity is explained by Figure 5. In that figure, if we propagate signal without using the translator gadget, we arrive at a configuration that is saturated but has only one fewer complex than a stable configuration. However, such near-stable configurations are still very different from the stable configuration of $\widetilde{T}_{n,k}$, so it is still possible to distinguish the two TBNs with an amplification factor proportional to 2^n and a resilience to false positives and negatives proportional to k.

The proof of this theorem is very similar to that of Theorem 3.1, and is also left to the appendix.

[2] The binding graph of the monomers within this giant polymer contains many complete k-ary trees of depth n as subgraphs. If each of the nodes of this graph is a real molecule that takes up some volume, it will be impossible to embed the whole graph within 3-dimensional space as n grows.

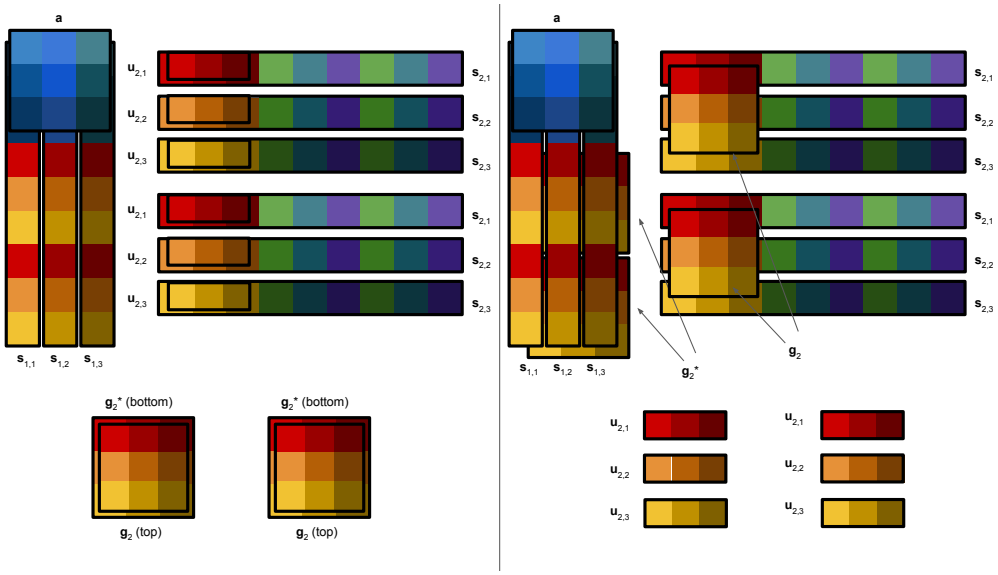


Figure 4 The amplifying translator gadget, before and after it is triggered to propagate the signal forward by one layer. When $\mathbf{s}_{1,1}$, $\mathbf{s}_{1,2}$ and $\mathbf{s}_{1,3}$ have been brought together, instead of directly replacing all the $\mathbf{u}_{2,j}$ monomers, they can split two $\{\mathbf{g}_2, \mathbf{g}_2^*\}$ complexes, and the $\mathbf{g}_2$ monomers can replace the $\mathbf{u}_{2,j}$ monomers.

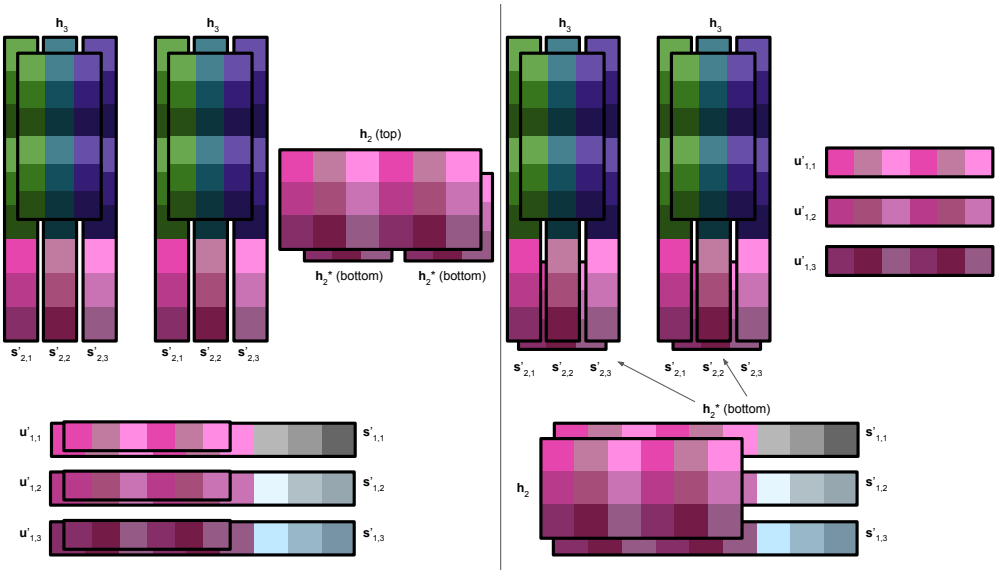


Figure 5 The converging translator gadget, before and after it is triggered to propagate the signal forward by one layer. When two copies each of $\mathbf{s}'_{2,1}$, $\mathbf{s}'_{2,2}$ and $\mathbf{s}'_{2,3}$ have been brought together into two complexes, instead of directly replacing all the $\mathbf{u}'_{1,j}$ monomers, they can split a $\{\mathbf{h}_2, \mathbf{h}_2^*, \mathbf{h}_2^*\}$ complex, and the $\mathbf{h}_2$ monomer can replace the $\mathbf{u}'_{1,j}$ monomers.
Note that in this image, the only way to propagate the signal efficiently would be to use the translator gadget; not using it will be one unit of entropy less efficient, requiring 3 splits and 4 merges, showing that this TBN no longer has an entropy gap. If instead we hadn't used the translator gadgets in the previous layer, then all six $\mathbf{s}'_{2,j}$ monomers in this image would be together in a single complex rather than on two separate complexes, in which case it would be equally efficient to either use this translator gadget or directly displace the $\mathbf{u}'_{1,j}$.

4 Upper Limit on TBN Signal Amplification

In this section, we show the following theorem providing an upper bound on the distance between a TBN before and after adding a single copy of a monomer, showing that the distance is at most double-exponential in the "size" of the system:

▶ **Theorem 4.1.** *Let T be a TBN with d domain types, m monomer types, and at most a domains on each monomer. Let $n = \max\{d, m, a\}$. Let T' be T with one extra copy of some monomer. Then $d(T, T') \leq n^{8n^{7n^2}}$.*

Recall Definition 2.7 for the distance between TBNs. Essentially, this theorem is saying that adding a single copy of some monomer can only impact doubly exponentially many total polymers, no matter how many total copies of each monomer are in the TBN.

Our strategy for proving this theorem is to fix some ordering on polymer types, and bound the distance between the lexicographically earliest stable configuration of an arbitrary TBN under that ordering before and after adding a single copy of some monomer. To bound this distance, we cast the problem of finding stable configurations of a TBN as an integer program (IP), and use methods from the theory of integer programming value functions to give a bound on how much the solution to this IP can change given a small change in the underlying TBN.

Proof. A complete proof including all technical details can found in the full version of this paper on arxiv. Here we present only the main ideas of the proof. ◀

We first introduce a definition from [6] and some notation that was unnecessary in previous sections.

▶ **Definition 4.2.** *Given a (star-limiting) TBN T, the* polymer basis *of T, denoted $\mathcal{B}(T)$, is the set of polymers $\mathbf{P}$ such that both of the following hold:*

- $\mathbf{P}$ *appears in some saturated configuration of a star-limiting TBN using the same monomer types as T.*
- $\mathbf{P}$ *cannot be split into two or more self-saturated polymers.*

The polymer basis is a useful construction because it is known to describe exactly those polymer types that may appear in stable configurations of T. It is always finite, and we will bound its size later.

Given a TBN T, let $M(T)$ denote its monomer types, and let $T(\mathbf{m})$ denote the count of monomer $\mathbf{m}$ in T. Given a polymer $\mathbf{P}$ and a monomer type $\mathbf{m} \in M(T)$, let $\mathbf{P}(\mathbf{m})$ represent the count of monomer $\mathbf{m}$ in polymer $\mathbf{P}$.

Suppose for the rest of this section that we have a TBN T to which we wish to add a single copy of some monomer $\mathbf{a}$ (which may or may not exist in T). Let T' be T with $\mathbf{a}$ added.

4.1 Finding Stable Configurations via Integer Programming

Prior work [6] has shown that the problem of finding the stable configurations of a TBN can be cast as an IP. There are multiple different formulations; we will use a formulation that is better for the purpose of reasoning theoretically about TBN behavior.

Let $\{x_{\mathbf{P}} : \mathbf{P} \in \mathcal{B}(T)\}$ be variables each representing the count of polymer $\mathbf{P}$ in a configuration of T. Then consider the following integer programming problem:

$$\begin{aligned} \max \quad & \sum_{\mathbf{P} \in \mathcal{B}(T)} x_{\mathbf{P}} \\ \text{s.t.} \quad & \sum_{\mathbf{P} \in \mathcal{B}(T)} \mathbf{P}(\mathbf{m}) x_{\mathbf{P}} = T(\mathbf{m}) \qquad \forall \mathbf{m} \in M(T) \\ & x_{\mathbf{P}} \in \mathbb{N} \qquad \forall \mathbf{P} \in \mathcal{B}(T) \end{aligned} \tag{2}$$

Intuitively, the linear equality constraints above express "monomer conservation": the total count of each monomer in T should equal the total number of times it appears among all polymers.

The following was shown in [6]; for the sake of self-containment, we show it here as well

▶ **Proposition 4.3.** *The optimal solutions to the IP* (2) *correspond exactly to stable configurations of* T*.*

Proof. If the variables $x_{\mathbf{P}}$ form a feasible solution, then those counts of polymers are a valid configuration because they exactly use up all monomers. If the solution is optimal, then there is no saturated configuration with more polymers (as only polymers from $\mathcal{B}(T)$ can show up in stable configurations), so the configuration is stable. Conversely, if a configuration σ is stable then it can be translated into a feasible solution to the IP because it only uses polymers from $\mathcal{B}(T)$ and obeys monomer conservation. If there were a solution with a greater objective function, then this would translate to a configuration with more complexes that is still saturated (because all polymers in the polymer basis are self-saturated), contradicting the assumption of σ's stability. ◀

We observe that adding an extra copy of some monomer to a TBN corresponds to changing the right-hand side of one of the constraints of this IP by one. Note that this is true even if we add a copy of some monomer for which there were 0 copies, as we may still include variables for polymers that contain that monomer in the former IP and simply consider there to be 0 copies of the monomer. Therefore, we are interested in sensitivity analysis of how quickly a solution to an IP can change as the right-hand sides of constraints change.

However, there is one edge case we must account for first. It is possible that T and T' have different polymer bases. This is because of the first requirement in Definition 4.2 requiring that the polymer basis respects that starred sites are limiting. If we add a single copy of a monomer, this may change which sites are limiting, if $\mathbf{a}$ has more copies of a starred site than T had excess copies of the unstarred site. We cannot include variables for such polymers in the IP formulation without taking extra precautions, as if we do there may be optimal solutions that don't correspond to saturated configurations. Therefore, we will first account for how many copies of such a polymer T and T' may differ by:

▶ **Lemma 4.4.** *Suppose that some polymer* $\mathbf{P}$ *is exactly one of* $\mathcal{B}(T')$ *and* $\mathcal{B}(T)$*. Then any saturated configuration of* T' *contains at most* $|\mathbf{a}|$ *copies of* $\mathbf{P}$*, where* $|\mathbf{a}|$ *denotes the number of sites on* $\mathbf{a}$*.*

Note that this result is slightly surprising – one natural way that one might try to design a TBN that amplifies signal is by designing the analyte so that it intentionally flips which sites are limiting. This result shows that this is an ineffective strategy: going from 5 excess copies of some site a to 5 excess copies of a^* is seemingly no more helpful in instigating a large change than going from 60 excess copies of a to 50.

Proof. If $\mathbf{P}$ is in $\mathcal{B}(T')$ but not $\mathcal{B}(T)$, it must contain an excess of a starred site that was limiting in T, but is no longer limiting in T'. We see this because $\mathbf{P}$ necessarily occurs in a saturated configuration of the TBN containing precisely the monomers that it is composed of; therefore, in order to not be in $\mathcal{B}(T)$, by definition of the polymer basis, it must be the case that this TBN has different limiting sites than T'.

Let a denote some such site type, so that a^* is limiting in T and a is limiting in T', and $\mathbf{P}$ contains an excess of a^*. Then $\mathbf{a}$ must contain an excess of a^*, but it cannot contain more than $|\mathbf{a}|$ excess copies. Therefore, there are at most this many *total* excess copies of a^* in T'. It follows that if there are more than $|\mathbf{a}|$ copies of $\mathbf{P}$ in a configuration of T', then those copies of $\mathbf{P}$ collectively have more excess copies of a^* than T' does, so some other polymer in that configuration would have to have an excess of a. This implies that such a configuration is not saturated (and therefore also cannot be stable). An identical argument shows that the same is true for polymers in $\mathcal{B}(T)$ but not $\mathcal{B}(T')$. ◀

In order to analyze and compare the two IP instances, we need them to have the same variable set. Therefore, we will include variables for all polymers from both polymer bases in both IP formulations. Let $\mathcal{B}(T, T') = \mathcal{B}(T) \cup \mathcal{B}(T')$ denote this merged polymer basis, and let $P = |\mathcal{B}(T, T')|$ denote the total number of possible polymers we must consider, or equivalently the number of variables we will have in these IPs. In each IP, we will have a constraint on each variable representing a polymer not in the relevant polymer basis, that says that that variable must equal zero.

4.2 Sensitivity Analysis

This sensitivity analysis problem of how IPs change as the right-hand sides of constraints change was studied by Blair and Jeroslow in [1]. We will not need their full theory, but we will use some of their results and methods.

In Corollary 4.7 of [1], they show that there is a constant K_3, independent of the right-hand sides of constraints (in our case, independent of how many copies of each monomer exist) such that:

$$R_c(v) \leq G_c(v) \leq R_c(v) + K_3, \tag{3}$$

where $G_c(v)$ gives the optimal value of the objective function c of a minimization IP as a function of the vector v of right-hand sides of constraints, and $R_c(v)$ gives the optimal value of the same problem when relaxing the constraint that variables must have integer values. The objective function we've shown so far is to maximize the sum of polymer counts rather than minimize, but the same statement applies that the integer and real-valued optimal solutions differ by at most K_3. In defining K_3, they also show the existence of a constant M_1 such that

$$|R_c(v) - R_c(w)| \leq M_1 \, ||v - w|| \,, \tag{4}$$

where v and w are different vectors for the right-hand sides of constraints. Note that we take all norms as 1-norms. Combining these inequalities, we see that

$$|G_c(v) - G_c(w)| \leq M_1 \, ||v - w|| + K_3. \tag{5}$$

For example, if we want to know the difference between the total number of polymers in a stable configuration before and after adding one copy of a monomer (and if the polymer bases of T and T' are identical), then we care about increasing one element of v by 1, so our bound on this difference is $M_1 + K_3$. This statement applies to maximization and minimization problems.

4.3 From Optimal Values to Polymer Counts

For ease of analysis, we order the polymers in $\mathcal{B}(T, T')$ as follows: first we list all the polymers that are not in $\mathcal{B}(T)$, then all the polymers that are not in $\mathcal{B}(T')$, then all the polymers in $\mathcal{B}(T) \cap \mathcal{B}(T')$. We need to show that the number of copies of each individual polymer does not change too much. We do this using a technique similar to Corollary 5.10 in [1].

Let P_{tot} be the total number of polymers in a stable configuration (either before or after adding $\mathbf{a}$, depending on which case we are examining).

We now define a new sequence of integer programs whose optimal values give polymer counts in the lexicographically earliest stable configuration under this ordering. We do this by finding the value of each variable $x_{\mathbf{P}}$ in order. This sequence of IP problems is defined separately for both TBNs, before and after adding $\mathbf{a}$.

For those variables representing a polymer that is in one basis but not the other, we do not need to analyze this IP, so we simply fix such a variable's value to whatever its value is in this lexicographically earliest stable configuration, which will be 0 in one TBN and bounded by $|\mathbf{a}|$ by Lemma 4.4 in the other.

Now, to find the value of some particular variable $x_{\mathbf{Q}}$ in either of the two TBNs where $\mathbf{Q} \in \mathcal{B}(T) \cap \mathcal{B}(T')$, suppose we have already found the value $y_{\mathbf{P}}$ we wish to fix $x_{\mathbf{P}}$ to for each $\mathbf{P} < \mathbf{Q}$ under our ordering. Then we define a new IP on all the same variables as follows:

$$
\begin{aligned}
\min \quad & x_{\mathbf{Q}} \\
\text{s.t.} \quad & \sum_{\mathbf{P} \in \mathcal{B}(T,T')} x_{\mathbf{P}} = P_{tot} \\
& \sum_{\mathbf{P} \in \mathcal{B}(T,T')} \mathbf{P}(\mathbf{m}) x_{\mathbf{P}} = T(\mathbf{m}) && \forall \mathbf{m} \in m(T) \\
& x_{\mathbf{P}} = y_{\mathbf{P}} && \forall \mathbf{P} < \mathbf{Q} \\
& x_{\mathbf{P}} \in \mathbb{N} && \forall \mathbf{P} \in \mathcal{B}(T, T')
\end{aligned} \tag{6}
$$

By construction, this IP gives us the smallest possible value that $x_{\mathbf{Q}}$ can take on in a stable configuration (as all variables must sum to P_{tot}) in which all previous $x_{\mathbf{P}}$ have fixed values. Then this process gives us a sequence of P (minus however many polymers were only in one polymer basis) different pairs of IP problems that we can sequentially compare to bound the differences between the values of the individual polymer counts in these lexicographically earliest configurations. We can repeatedly apply Equation (5) to each $x_{\mathbf{P}}$ in turn, as each variable's value before and after adding $\mathbf{a}$ will be given by the optimal value of (6) where the only differences are in the right-hand sides of constraints.

5 Conclusion

In this paper we have defined the signal amplification problem for Thermodynamic Binding Networks, and we have demonstrated a TBN that achieves exponential signal amplification. We also showed a doubly-exponential upper bound for the problem. As TBNs model mixtures of DNA, a TBN that amplifies signal can potentially be implemented as a real system. An upper bound has implications for how effective a system designed in this way can potentially be, and shows that there are some limitations for a purely thermodynamic approach to signal detection and amplification.

One clear direction for future work is to implement such a system. This would involve creating a design that accounts for the simplifications of the TBN model. In particular, enthalpy and entropy need to be strong enough with enthalpy sufficiently stronger than

entropy. Further, the polymers formed need to be geometrically feasible. We have done some work to make this problem geometrically realizable with the inclusion of translator gadgets in Section 3.3. In principle, the polymers that are formed in this version of the system are simple enough that they should form if the DNA strands implementing them are well-designed.

Another goal would be to bridge the gap between our singly exponential amplifier and doubly exponential upper bound by either describing a TBN that can amplify signal more than exponentially, or deriving a more precise upper bound. If one wished to construct a TBN with doubly exponential amplification, an examination of our upper bound proof will show that such a TBN must have an exponentially sized polymer basis, and most likely would need to actually use an exponential amount of different polymer types in its stable configurations either with or without the analyte. Such a design seems relatively unlikely to come to fruition, and it seems more likely that our proof technique or similar techniques can be tightened in order to show a stricter upper bound. Thus, we conjecture that the true upper bound is (singly) exponential.

There are also other types of robustness that we have not discussed in this work that merit further analysis. One of these is input specificity: the question of how well the system amplifies signal if the analyte is changed slightly. Another is sensitivity to the number of copies of each component. Intuitively, our system's behavior depends on having exactly equal numbers of complementary strands within each layer; if there are too many copies of one, it may result in those excess copies spuriously propagating or blocking signal to the next layer. This issue may be intrinsic to thermodynamic signal amplifiers, or there may be some system more robust to it. Lastly, it may be experimentally useful to show that our system achieves its stable states not only in the limit of thermodynamic equilibrium, but also more practically when annealed. Some systems such as HCR are designed to reach non-equilibrium, meta-stable states when annealed. We conjecture that our system should reach equilibrium when annealed, because kinetic traps in the system are far away from being thermodynamically stable (large entropy gap). Formally studying annealing could be done by analyzing versions of the TBN model with different tradeoffs between entropy and enthalpy to model different temperatures.

References

1 C. E. Blair and R. G. Jeroslow. The value function of an integer program. *Mathematical Programming*, 23(1):237–273, December 1982. doi:10.1007/BF01583794.

2 Keenan Breik, Cameron Chalk, David Doty, David Haley, and David Soloveichik. Programming substrate-independent kinetic barriers with thermodynamic binding networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 18(1):283–295, 2021. doi:10.1109/TCBB.2019.2959310.

3 Keenan Breik, Chris Thachuk, Marijn Heule, and David Soloveichik. Computing properties of stable configurations of thermodynamic binding networks. *Theoretical Computer Science*, 785:17–29, 2019.

4 Harry MT Choi, Maayan Schwarzkopf, Mark E Fornace, Aneesh Acharya, Georgios Artavanis, Johannes Stegmaier, Alexandre Cunha, and Niles A Pierce. Third-generation in situ hybridization chain reaction: Multiplexed, quantitative, sensitive, versatile, robust. *Development*, 145(12):dev165753, 2018.

5 David Doty, Trent A. Rogers, David Soloveichik, Chris Thachuk, and Damien Woods. Thermodynamic binding networks. In Robert Brijder and Lulu Qian, editors, *DNA Computing and Molecular Programming*, pages 249–266, Cham, 2017. Springer International Publishing.

6 David Haley and David Doty. Computing properties of thermodynamic binding networks: An integer programming approach. In Matthew R. Lakin and Petr Šulc, editors, *DNA 2021: Proceedings of the 27th International Meeting on DNA Computing and Molecular Programming*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DNA.27.2.

7 Randolph Lopez, Ruofan Wang, and Georg Seelig. A molecular multi-gene classifier for disease diagnostics. *Nature chemistry*, 10(7):746–754, 2018.

8 Dionis Minev, Christopher M Wintersinger, Anastasia Ershova, and William M Shih. Robust nucleation control via crisscross polymerization of highly coordinated DNA slats. *Nature communications*, 12(1):1741, 2021.

9 Gerald Schochetman, Chin-Yih Ou, and Wanda K. Jones. Polymerase chain reaction. *The Journal of Infectious Diseases*, 158(6):1154–1157, 1988. URL: http://www.jstor.org/stable/30137034.

10 Erhu Xiong, Dongbao Yao, Andrew D. Ellington, and Sanchita Bhadra. Minimizing leakage in stacked strand exchange amplification circuits. *ACS Synthetic Biology*, 10(6):1277–1283, 2021. PMID: 34006090. doi:10.1021/acssynbio.0c00615.

A Proofs for Section 3 (Signal Amplification TBN)

▶ **Lemma 3.2.** *$T_{n,k}$ has exactly one stable configuration $\sigma_{n,k}$.*

Proof. We consider merges to get from the melted configuration to any saturated configuration. We may order these merges such that we first make all the merges necessary to cover each individual $\mathbf{s}_{i,j}$ in increasing value of i, then each individual $\mathbf{s}'_{i,j}$ in decreasing value of i. We see that at each step of this process, we may cover the monomer in question by a single merge (of its corresponding $\mathbf{u}_{i,j}$ or $\mathbf{u}'_{i,j}$). If we never merge the corresponding $\mathbf{u}$ monomer, the only other monomers that can cover the starred sites on a given $\mathbf{s}_{i,j}$ are k different $\mathbf{s}_{i-1,\ell}$ monomers. Likewise, the only other way to cover the starred sites on a given $\mathbf{s}'_{i,j}$ is by using k different $\mathbf{s}'_{i+1,\ell}$ monomers (except for $\mathbf{s}'_{n,j}$ which needs $\mathbf{s}_{n,\ell}$ monomers).

If an $\mathbf{s}$ or $\mathbf{s}'$ monomer is covered in multiple different ways, we order the merges such that it is first covered by one corresponding $\mathbf{u}$ monomer (and then ignore any other merges for now, as we are still ordering the merges to cover each $\mathbf{s}$ monomer sequentially). We see then that if every $\mathbf{s}$ monomer is covered by a $\mathbf{u}$ monomer, then no $\mathbf{s}$ monomers will be brought together during this process. Therefore, the first time in this sequence that we choose to cover an $\mathbf{s}$ without its corresponding $\mathbf{u}$ will require k total merges to cover that $\mathbf{s}$. The resulting configuration is feed-forward, so by Corollary 2.11, reaching a stable configuration requires at least one more merge per remaining $\mathbf{s}$ monomer. This results in at least $k-1$ extra merges compared to covering $\mathbf{s}$ and $\mathbf{s}'$ monomers by using $\mathbf{u}$ and $\mathbf{u}'$ monomers respectively.

Once all $\mathbf{s}$ and $\mathbf{s}'$ monomers are covered, the only other monomer with starred sites is $\mathbf{p}^*$, so we can make all the merges that are needed to cover it. If none of the $\mathbf{s}'_{1,j}$ monomers have been brought together, then the fewest merges it takes to cover $\mathbf{p}^*$ is $\frac{k}{2}$, via the $\mathbf{p}_j$ monomers. If any of them have been brought together, then it could potentially take a single merge to cover $\mathbf{p}^*$. However, this would have required $k-1$ extra merges at some point during the covering of $\mathbf{s}$ monomers, resulting in $\frac{k}{2}$ extra total merges compared to covering all $\mathbf{s}$ monomers with $\mathbf{u}$ monomers, then covering $\mathbf{p}^*$ with $\mathbf{p}_j$ monomers.

Therefore, this latter set of merges covers all starred sites in as few merges as possible, and therefore gives the unique stable configuration of $T_{n,k}$.

◀

▶ **Corollary 3.3.** *$T_{n,k}$ has an entropy gap of $\frac{k}{2}-1$.*

Proof. Recall Definition 2.8 for what we must show. Any saturated configuration that does not make all the merges in $\sigma_{n,k}$ must either have some $\mathbf{s}$ that is not covered by its corresponding $\mathbf{u}$ (resulting in at least $\frac{k}{2}$ extra merges, as per the above argument), or must cover $\mathbf{p}^*$ with initially-separate $\mathbf{s}'_{1,j}$ monomers (resulting in $\frac{k}{2}$ extra merges). Thus, any such configuration has distance to stability at least $\frac{k}{2}$. Any other saturated configuration that does make all of the merges in this sequence simply makes some extra merges afterward, and therefore splits to $\sigma_{n,k}$. It follows that $T_{n,k}$ has an entropy gap of $\frac{k}{2}$ (and also of $\frac{k}{2}-1$, for consistency in the statement of Theorem 3.1). ◀

▶ **Lemma 3.4.** *$T^{\mathrm{a}}_{n,k}$ has exactly one stable configuration $\sigma^{\mathrm{a}}_{n,k}$, and $T^{\mathrm{a}}_{n,k}$ has an entropy gap of $\frac{k}{2}-1$.*

Proof. We see that $T^{\mathrm{a}}_{n,k}$ (like $T_{n,k}$) is feed-forward (recall Definition 2.9) by first ordering $\mathbf{a}$ along with all the $\mathbf{u}_{i,j}$, $\mathbf{u}'_{i,j}$, and $\mathbf{p}_j$ monomers (none of which have starred sites), then all the $\mathbf{s}_{i,j}$ in increasing order of i, then all the $\mathbf{s}'_{i,j}$ in decreasing order of i, and finally $\mathbf{p}^*$.

Unlike $T_{n,k}$, however, we may reach a stable state by merging $\mathbf{a}$ together with every single $\mathbf{s}_{i,j}$, every single $\mathbf{s}'_{i,j}$ and $\mathbf{p}^*$ into a single polymer. This covers all starred sites, and requires

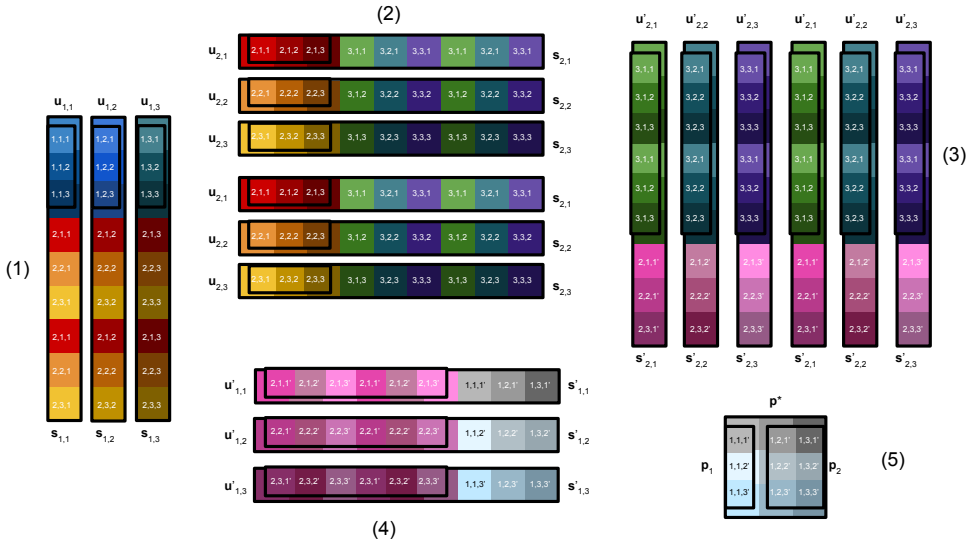


Figure 6 A version of Figure 2 with text labels on domains, for accessibility and allowing comparison with the domains as defined in the text of the paper. This figure shows the unique stable configuration of $T_{2,3}$.

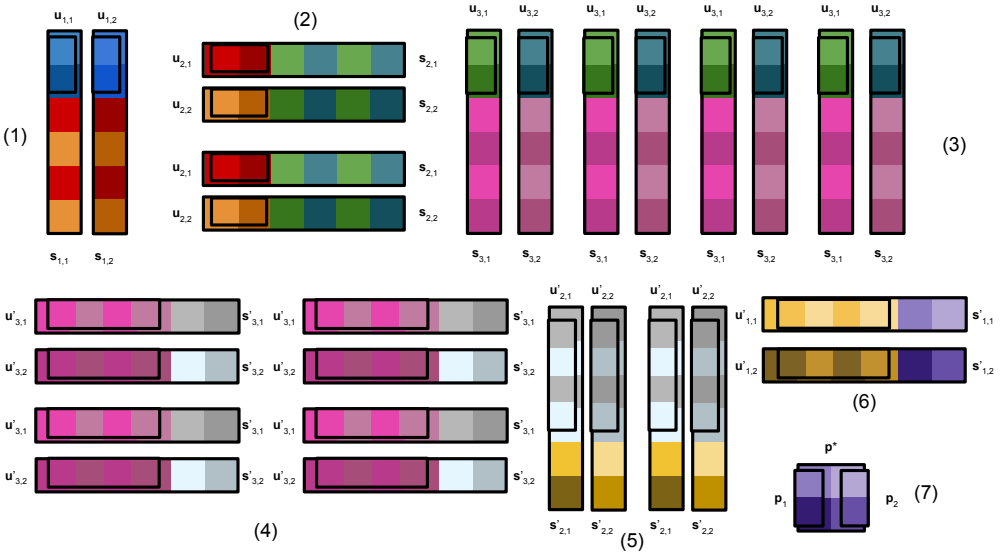


Figure 7 The unique stable configuration $\sigma_{3,2}$ of $T_{3,2}$. Compared to Figure 2 (or Figure 6 above), which show $T_{2,3}$, this figure shows one more layer and a smaller entropy gap parameter. The additional layer means that there are 4 copies of each monomer in the largest parts of the figure, compared to 2 copies of each monomer in the other figures; if another layer were added, it would contain 8 copies of each monomer. The smaller entropy gap parameter manifests in this figure visually having a "2 by 2 grid" design motif, compared to the "3 by 3 grid" motif in the other figures.

exactly one merge per monomer with starred sites, so by Corollary 2.12, this configuration $\sigma^{\mathrm{a}}_{n,k}$ is stable.

Now, we examine an arbitrary saturated configuration σ of $T^{\mathrm{a}}_{n,k}$. We consider merges in essentially the opposite order of how they were considered when analyzing $T_{n,k}$. First, consider $\mathbf{p}^*$. It must be covered either by all the $\mathbf{p}_j$ monomers, or by all the $\mathbf{s}'_{1,j}$ monomers. If we merge all the $\mathbf{p}_j$ monomers to $\mathbf{p}^*$, we arrive at a configuration that is still feed-forward, but has only one fewer polymer with uncovered starred sites compared to $\mathrm{melt}(T^{\mathrm{a}}_{n,k})$ in spite of making $\frac{k}{2}$ merges. Therefore, by Lemma 2.10, reaching a saturated configuration from this point requires at least $\frac{k}{2}-1$ extra merges compared to $\sigma^{\mathrm{a}}_{n,k}$.

Now, we may make a similar argument for all $\mathbf{s}$ monomers in the opposite order that we considered them in Lemma 3.2. First, either we have already made $\frac{k}{2}-1$ extra merges, or the $\mathbf{s}'_{1,j}$ monomers have all been brought together on a single polymer to cover $\mathbf{p}^*$. If we now make all the merges necessary to cover all starred sites on this polymer, we must do so either using all the $\mathbf{u}'_{1,j}$ or by using all the $\mathbf{s}'_{2,j}$. If we use the former, then this will require k total merges but will only reduce the count of polymers with starred sites by 1. The resulting configuration is still feed-forward, so again by Lemma 2.10 any saturated configuration we reach from this point will require at least $k-1$ extra merges compared to σ. Otherwise, we must bring all the $\mathbf{s}_{2,j}$ monomers together to cover these sites. This does not fall victim to the same argument, because bringing these monomers with starred sites together onto the same polymer lowers the total number of polymers with uncovered starred sites. Now that they have been brought together, the same argument shows that we must either cover all the starred sites on the $\mathbf{s}'_{2,j}$ using all the $\mathbf{s}'_{3,j}$, or suffer $k-1$ extra merges. The same argument for each layer in the converging part of the TBN also works for each layer in the amplifying part. Finally, after running through this argument we arrive at all $\mathbf{s}_{1,j}$ being brought together, which can be covered either by a single merge of $\mathbf{a}$ or by merging the k $\mathbf{u}_{1,j}$ to it.

Overall, this shows that any saturated configuration of $T^{\mathrm{a}}_{n,k}$ either makes all of the merges in $\sigma^{\mathrm{a}}_{n,k}$ or it must make at least $\frac{k}{2}-1$ extra merges. It follows that $\sigma^{\mathrm{a}}_{n,k}$ is the unique stable configuration of $T^{\mathrm{a}}_{n,k}$, with an entropy gap of $\frac{k}{2}-1$ as desired. ◀

▶ **Theorem 3.5.** *Let $\widetilde{T} = \widetilde{T}_{n,k}$ and $\widetilde{T}^{\mathrm{a}} = \widetilde{T}^{\mathrm{a}}_{n,k}$ be as described. Then:*

1. *$\widetilde{T}$ has exactly one stable configuration $\widetilde{\sigma}_{n,k}$, and $d(\widetilde{T}, \widetilde{T}^{\mathrm{a}}) > 2^n$.*
2. *$\widetilde{T}$ has an entropy gap of $\frac{k}{2}$, and $\widetilde{T}^{\mathrm{a}}$ has the property that all of its configurations α that are within distance to stability $\frac{k}{2}$ satisfy $d(\widetilde{\sigma}_{n,k}, \alpha) > 2^n$.*
3. *$\widetilde{T} = \widetilde{T}_{n,k}$ uses $\mathcal{O}(nk)$ total monomer types, $\mathcal{O}(nk^2)$ domain types, and $\mathcal{O}(k^2)$ domains per monomer.*
4. *The unique stable configuration of $\widetilde{T}$ has $\mathcal{O}(k)$ monomers in its largest polymer. There is a stable configuration of $\widetilde{T}^{\mathrm{a}}$ sharing this property.*

Proof. Recall the constructions of $\widetilde{T}_{n,k}$ and $\widetilde{T}^{\mathrm{a}}_{n,k}$ from Section 3.3. Our argument will be very similar to that of Theorem 3.1 (i.e., the above lemmas), except we need to account for the extra monomer types.

First, consider $\widetilde{T}_{n,k}$, where $\mathbf{a}$ is absent. We wish to show that its stable configuration looks like that of $T_{n,k}$, with the added $\mathbf{g}$ and $\mathbf{h}$ monomers only binding to added $\mathbf{g}^*$ and $\mathbf{h}^*$ monomers respectively. We order the merges to get to a saturated configuration in essentially the same order as we did in analyzing $T_{n,k}$: first we will make all merges necessary to cover all $(1,j,\ell)^*$ sites, then $(2,j,\ell)^*$, and so on up to $(n+1,j,\ell)^*$, then $(n,j,\ell)'^*$, and so on. As before, at each step, we will see that we cannot make merges in any way other than those in the desired stable configuration without needing $k-1$ extra merges for that step.

For $(i,j,\ell)^*$ sites, at each step, there is exactly one way to cover all starred sites by making one merge per monomer with these starred sites: we cover each $\mathbf{s}_{i,j}$ with a $\mathbf{u}_{i,j}$ and each $\mathbf{g}_i^*$ with a $\mathbf{g}_i$. In particular, we already know from the proof of Lemma 3.2 that this is true for the $\mathbf{s}_{i,j}$ if we only use $\mathbf{s}_{i-1,j}$ and $\mathbf{u}_{i,j}$ to cover it, and that we will otherwise need to make $k-1$ extra merges. Clearly we also cannot cover $\mathbf{g}_i^*$ with anything other than $\mathbf{g}_i$ without making k merges to cover it (and thus $k-1$ extra merges), so we cannot use $\mathbf{g}_i$ to cover $\mathbf{s}_{i,j}$.

Likewise, for $(i,j,\ell)'^*$ sites, we only need to observe that each $\mathbf{h}_i^*$ monomer can only be covered in a single merge by $\mathbf{h}_i$, so any other way of making merges necessarily involves $k-1$ extra merges. So by the same argument as in Lemma 3.2 and Corollary 3.3, $\widetilde{T}_{n,k}$ has exactly one stable configuration with an entropy gap of $\frac{k}{2}$. This configuration has $1+\frac{k}{2}$ monomers in the polymer containing $\mathbf{p}^*$ and all the $\mathbf{p}_j$, 3 monomers in each $\{\mathbf{h}_i, \mathbf{h}_i^*, \mathbf{h}_i^*\}$ polymer, and 2 monomers in each other polymer.

Now, consider $\widetilde{T}^{\mathrm{a}}_{n,k}$, where $\mathbf{a}$ is present. If we take the stable configuration of $T^{\mathrm{a}}_{n,k}$ and simply put all $\mathbf{g}$ monomers into $\{\mathbf{g}_i, \mathbf{g}_i^*\}$ polymers, and all the $\mathbf{h}$ monomers into $\{\mathbf{h}_i, \mathbf{h}_i^*, \mathbf{h}_i^*\}$ polymers, we have still made exactly one merge per monomer with any starred sites, so by Corollary 2.12 it is stable. If we then carry out the shifts described in Figure 4 and Figure 5, an equal number of merges and splits are made at each step, so the resulting saturated configuration is still stable. Additionally, in this configuration, the largest polymers have $k+3$ monomers (specifically, those containing a set of $\mathbf{s}_{i,j}$ along with one copy of $\mathbf{g}_i$ and two copies of $\mathbf{g}_{i+1}^*$).

All that remains to show is that all configurations of $\widetilde{T}^{\mathrm{a}}_{n,k}$ that are within $\frac{k}{2}$ distance to stability have exponentially many different polymers from the stable configuration of $\widetilde{T}_{n,k}$. We will do this by showing that all $\mathbf{u}$ and $\mathbf{u}'$ monomers are free in all such configurations.

Again, this argument is very similar to the argument without the translator gadgets in Lemma 3.4. We consider merges to cover starred sites in the opposite order of the above argument for $\widetilde{T}_{n,k}$. First, consider the merges necessary to cover all the $(1,j,\ell)'$ starred sites (on $\mathbf{p}^*$). Like before, they must be covered by either all the $\mathbf{u}'_{1,j}$ monomers or all the $\mathbf{p}_j$ monomers, but using the latter gives a feed-forward configuration in which $\frac{k}{2}-1$ extra merges have already been made. Thus, to be within $\frac{k}{2}$ distance to stability, we must use the $\mathbf{s}'_{1,j}$. Next, for the $(2,j,\ell)'$ starred sites, with the merges already made, there are two copies of each of these sites all together on the polymer containing all the $\mathbf{s}'_{1,j}$, and one copy of each site on each of the two $\mathbf{h}_2^*$ monomers. If we are to merge any $\mathbf{u}'_{1,j}$ monomers to any of these in such a way that they cannot be split off without the result still being saturated, then we must merge all of the $\mathbf{u}'_{1,j}$ into one polymer. Like with the argument for $\widetilde{T}^{\mathrm{a}}_{n,k}$, we see that this results in $k-1$ extra merges compared to a stable configuration. Thus, we cannot use any $\mathbf{u}'_{1,j}$, and these sites must be covered by the $\mathbf{s}'_{2,j}$ and $\mathbf{h}_2$ monomers.

We may do this either by using the $\mathbf{h}_2$ to cover both $\mathbf{h}_2^*$ (in effect, not using the translator gadget) or by using $\mathbf{h}_2$ to cover all the $\mathbf{s}'_{1,j}$. The only difference in terms of the argument is that in the former case *all* of the $\mathbf{s}_{2,j}$ will be brought together in a single polymer, and in the latter case they will be split between two polymers. In the former case, it may require one extra merge to use translator gates in the next layer; however, either way, the same argument on each other layer in sequence shows that we cannot use any $\mathbf{u}'_{i,j}$ monomers without suffering $\frac{k}{2}-1$ extra merges. Likewise, the exact same argument shows that the same thing is true of $\mathbf{u}_{i,j}$ monomers, necessitating that in any configuration that makes fewer than $\frac{k}{2}-1$ extraneous merges, all exponentially many $\mathbf{u}$ and $\mathbf{u}'$ monomers must be free, as desired. ◀

Optimal Information Encoding in Chemical Reaction Networks

Austin Luchsinger
Electrical and Computer Engineering, University of Texas at Austin, TX, USA

David Doty
Computer Science, University of California–Davis, CA, USA

David Soloveichik
Electrical and Computer Engineering, University of Texas at Austin, TX, USA

Abstract

Discrete chemical reaction networks formalize the interactions of molecular species in a well-mixed solution as stochastic events. Given their basic mathematical and physical role, the computational power of chemical reaction networks has been widely studied in the molecular programming and distributed computing communities. While for Turing-universal systems there is a universal measure of optimal information encoding based on Kolmogorov complexity, chemical reaction networks are not Turing universal unless error and unbounded molecular counts are permitted. Nonetheless, here we show that the optimal number of reactions to generate a specific count $x \in \mathbb{N}$ with probability 1 is asymptotically equal to a "space-aware" version of the Kolmogorov complexity of x, defined as $\widetilde{\mathrm{Ks}}(x) = \min_p \{|p|/\log|p| + \log(\mathtt{space}(\mathcal{U}(p))) : \mathcal{U}(p) = x\}$, where p is a program for universal Turing machine $\mathcal{U}$. This version of Kolmogorov complexity incorporates not just the length of the shortest program for generating x, but also the space usage of that program. Probability 1 computation is captured by the standard notion of stable computation from distributed computing, but we limit our consideration to chemical reaction networks obeying a stronger constraint: they "know when they are done" in the sense that they produce a special species to indicate completion. As part of our results, we develop a module for encoding and unpacking any b bits of information via $O(b/\log b)$ reactions, which is information-theoretically optimal for incompressible information. Our work provides one answer to the question of how succinctly chemical self-organization can be encoded – in the sense of generating precise molecular counts of species as the desired state.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases chemical reaction networks, Kolmogorov complexity, stable computation

Digital Object Identifier 10.4230/LIPIcs.DNA.29.9

Funding *Austin Luchsinger*: Carroll H. Dunn Endowed Graduate Fellowship in Engineering.
David Doty: NSF grants 2211793, 1900931, and CAREER-1844976.
David Soloveichik: NSF grant 1901025, Sloan Foundation Research Fellowship.

Acknowledgements The authors thank Eric Severson for the invaluable discussions on chemical reaction network computation that helped lead to the results presented here.

1 Introduction

In potential biochemical, nanotechnological, or medical applications, synthetic chemical computation could allow for the re-programming of biological regulatory networks and the insertion of control modules where traditional electronic controllers are not feasible. Understanding the design principles of chemical information processing also may achieve better understanding of the complex information processing that occurs in biological chemical interactions.

Discrete chemical reaction networks, also called stochastic chemical reaction networks, is a formal model of chemical kinetics in a well-mixed solution. While in continuous chemical kinetics, continuous concentrations change in time governed by ordinary differential questions, here the state consists of non-negative integer molecular counts of the species, and reaction events occur stochastically as a continuous time Markov process. Closely related models include population protocols in distributed computing [4], as well as models without stochastic kinetics such as Petri nets [20], vector addition systems [22] and commutative semigroups [11]. The model is particularly relevant when some species are present in small molecular counts, which are not well-approximated by continuous concentrations [21]; this regime is germane for small volumes such as that of a cell, natural or artificial. For the rest of this paper, the acronym CRNs (Chemical Reaction Networks) refers to the discrete model.

Typically the ensuing sequence of reactions can be predicted only stochastically since multiple reactions compete with each other. Nonetheless certain behaviors are independent of the order in which reactions happen to occur. Such probability 1 behavior is formalized using the notion of stable computation. For example the reactions $X_1 \to 2Y$ and $X_2 + Y \to \emptyset$ compute the function $f(x_1, x_2) = \max(2x_1 - x_2, 0)$ regardless of the order in which reactions happen. Below when we say that a CRN computes something, we mean it in the sense of stable computation. It is known that stably computing CRNs are not Turing-universal [36], but instead are limited to computing semilinear predicates and functions [5, 13]. However, the scaling of the computational power of CRNs with the number of reactions and species still lacks a tight and general characterization.

Prior approaches to answering the question of reaction or species complexity – in the equivalent language of population protocols – have focused largely on predicate computation and can be divided into two groups. (We should point out that the literature makes the important distinction between population protocols with and without a "leader," which is equivalent to starting with a single copy of a distinguished species in the initial state. The prior results described here as well as our work correspond to protocols *with* a leader.) The first line of work focuses on specific predicates – with the prototypical choice being the so-called "counting predicates" in which the task is to decide whether the count of the input species is at least some threshold $x \in \mathbb{N}$ [8, 17, 26]. In particular, close upper and lower bounds were developed: for infinitely many x, the predicate can be stably decided with $\mathcal{O}(\log \log x)$ species [8], and $\mathcal{O}((\log \log x)^{1/2-\epsilon})$ species are required [26].

Other work has focused on the more general characterization of predicate computation. It is well-known that semilinear predicates can be characterized in terms of Presburger arithmetic, the first-order theory of addition. It was subsequently shown that a CRN can decide a semilinear predicate with the number of species scaling polynomially with the size of the corresponding Presburger formula [7, 18]. There are also provable tradeoffs between the speed of computation and the number of species (e.g., [2, 6, 19]). We do not consider the time-complexity of CRNs further in this paper.

While the prior work described above involves stably deciding a counting predicate where the system recognizes if the count of some species is at least x, we investigate the problem of generating exactly x copies of a particular species Y, starting from a single copy of another species L. This idea of generation is natural for engineers of these systems who may wish to prepare a particular configuration to be used in a downstream process, and captures a certain form of chemical self-organization. (We note the conceptual connection to another type of self-organization: leader-election, in which we want to end up with exactly one molecule of a species, starting from many [6].) Our constructions can be adopted to deciding the counting predicates with only a constant more reactions – giving a novel upper bound on the number

of reactions (see Open Questions). It is also worth noting that other complexity questions have been investigated for CRNs, such as "the size of the smallest chemical reaction network that approximates a desired distribution" [10].

The goal of this paper is to connect the complexity of the most compact CRN for generating x to the well-known measures of the optimal "description length" of x. Kolmogorov complexity, a widely recognized concept across various disciplines in computer science and information theory, serves as a universal, broadly accepted measure of description length [27]. This notion quantifies the complexity of an object, such as a string or a number, by the length of the shortest program that produces it. While the minimal number of species or reactions to generate count x cannot be connected to the canonical Kolmogorov complexity, we provide tight asymptotic bounds to a modification of Kolmogorov complexity $\widetilde{\mathrm{K}}\mathrm{s}$ (Equation (1)). As this quantity incorporates not only the length of the shortest program to produce x, but also the space (memory) usage of the program, it can be called "space-aware." Unlike the canonical Kolmogorov complexity, $\widetilde{\mathrm{K}}\mathrm{s}$ is computable.

Our quantity $\widetilde{\mathrm{K}}\mathrm{s}$ characterizes the CRN complexity of generating x in the range from $\mathcal{O}(\log\log x)$ for highly "compressible" x to $\mathcal{O}(\log x/\log\log x)$ for "incompressible" x. The module we develop for optimally encoding b bits of information with $\mathcal{O}(b/\log b)$ reactions via a permutation code may be of independent interest. The encoded information could be used for other purposes than for generating a desired amount of some species, which justifies a more general interpretation of our work as studying the encoding information in CRNs.

2 Preliminaries

We use notation from [12, 35] and stable computation definitions from [5, 14] for (discrete) chemical reaction networks. Let $\mathbb{N}$ denote the nonnegative integers. For any finite set $\mathcal{S}$ (of species), we write $\mathbb{N}^{\mathcal{S}}$ to mean the set of functions $f : \mathcal{S} \to \mathbb{N}$. Equivalently, $\mathbb{N}^{\mathcal{S}}$ can be interpreted as the set of vectors indexed by the elements of $\mathcal{S}$, and so $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ specifies nonnegative integer counts for all elements of $\mathcal{S}$. For $\mathbf{a}, \mathbf{b} \in \mathbb{N}^{\mathcal{S}}$, we write $\mathbf{a} \leq \mathbf{b}$ if $\mathbf{a}(i) \leq \mathbf{b}(i), \forall i$.

2.1 Chemical Reaction Networks

A *chemical reaction network* (CRN) $\mathcal{C} = (\mathcal{S}, \mathcal{R})$ is defined by a finite set $\mathcal{S}$ of species, and a finite set $\mathcal{R}$ of reactions where each reaction is a pair $\langle \mathbf{r}, \mathbf{p} \rangle \in \mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}$ that denotes the *reactant* species consumed by the reaction and the *product* species generated by the reaction. For example, given $\mathcal{S} = \{A, B, C\}$, the reaction $\langle (2,0,0), (0,1,1) \rangle$ represents $2A \to B + C$. Although the definition allows for more general stoichiometry, in this paper we only consider third-order reactions (with at most three reactants and three products). For reversible reactions, we will use the notation $A + B \rightleftarrows C + D$ to mean $A + B \to C + D$ and $C + D \to A + B$. We say that the *size* of a CRN (denoted $|\mathcal{C}|$) is simply the number of reactions in $\mathcal{R}$.[1]

A *configuration* $\mathbf{c} \in \mathbb{N}^{\mathcal{S}}$ of a CRN assigns integer counts to every species $s \in \mathcal{S}$. When convenient, we use the notation $\{n_1 S_1, n_2 S_2, \ldots, n_k S_k\}$ to describe a configuration with $n_i \in \mathbb{N}$ copies of species $\mathcal{S}_i, \forall i \in [1, k]$. When using this notation, any species $S_j \in \mathcal{S}$ that is not listed is assumed to have a zero count (e.g., given $\mathcal{S} = \{A, B, C\}$, the configuration $\{3A, 2B\}$ has three copies of species A, two copies of species B, and zero of species C). For two configurations $\mathbf{a}, \mathbf{b} \in \mathbb{N}^{\mathcal{S}}$, we say $\mathbf{b}$ *covers* $\mathbf{a}$ if $\mathbf{a} \leq \mathbf{b}$; in other words, for all species, $\mathbf{b}$ has at least as many copies as $\mathbf{a}$.

[1] When considering systems with third-order reactions it is clear that $|\mathcal{R}|^{1/6} \leq |\mathcal{S}| \leq 6|\mathcal{R}|$.

A reaction $\langle \mathbf{r}, \mathbf{p} \rangle$ is said to be *applicable* in configuration $\mathbf{c}$ if $\mathbf{r} \leq \mathbf{c}$. If the reaction $\langle \mathbf{r}, \mathbf{p} \rangle$ is applicable, it results in configuration $\mathbf{c}' = \mathbf{c} - \mathbf{r} + \mathbf{p}$ if it occurs, and we write $\mathbf{c} \rightarrowtail \mathbf{c}'$. If there exists a finite sequence of configurations such that $\mathbf{c} \rightarrowtail \mathbf{c}_1 \rightarrowtail \ldots \rightarrowtail \mathbf{c}_n \rightarrowtail \mathbf{d}$, then we say that $\mathbf{d}$ is *reachable* from $\mathbf{c}$ and we write $\mathbf{c} \rightsquigarrow \mathbf{d}$.

In keeping with established definitions for stable computation, we specify an *output species* $Y \in \mathcal{S}$ and a *leader species* $L \in \mathcal{S}$ for stable integer computation.[2] We start from an initial configuration $\mathbf{i} = \{1L\}$. A configuration $\mathbf{c}$ is *output-stable* if $\forall \mathbf{d}$ such that $\mathbf{c} \rightsquigarrow \mathbf{d}$, $\mathbf{c}(Y) = \mathbf{d}(Y)$. CRN $\mathcal{C}$ *stably computes* integer x if, from any configuration $\mathbf{c}$ that is reachable from input configuration $\mathbf{i}$, there is an output-stable configuration $\mathbf{o}$ reachable from $\mathbf{c}$ with $\mathbf{o}(Y) = x$. Note that when considering systems with bounded state spaces like those discussed in this paper, stable computation is equivalent to probability 1 computing.

We also consider a much stronger constraint on CRN computation that specifies a special halting species. A species $H \in \mathcal{S}$ is a *halting species* if $\forall \mathbf{c}$ such that $\mathbf{c}(H) \geq 1$, $\mathbf{c}$ is output stable and $\forall \mathbf{d}$ where $\mathbf{c} \rightsquigarrow \mathbf{d}$, $\mathbf{d}(H) \geq 1$. We say that a CRN $\mathcal{C}$ *haltingly computes* an integer x if (1) $\mathcal{C}$ stably computes x and (2) $\mathcal{C}$ has a halting species H. Intuitively, a halting CRN knows when it is done – the halting species can initiate some downstream process that is only meant to occur when the computation is finished.

2.2 Kolmogorov Complexity

A focus of this paper is the "optimal description" of integers. As such, we often refer to the traditional notion of Kolmogorov complexity which we define here.

Let $\mathcal{U}$ be a universal Turing machine. The Kolmogorov complexity for an integer x is the value $\mathrm{K}(x) = \min\{|p| : \mathcal{U}(p) = x\}$. In other words, the Kolmogorov complexity of x is the size of the smallest Turing machine program p that outputs x. This captures the descriptional complexity of x in the sense that a (smaller) description of x can be given to some machine that generates x based on the given description.

We use a "space-aware" variant of this quantity which we later connect to the size of the smallest CRN stably computing x:

$$\widetilde{\mathrm{K}}\mathrm{s}(x) = \min\left\{ \frac{|p|}{\log|p|} + \log(\mathtt{space}(\mathcal{U}(p))) : \mathcal{U}(p) = x \right\}. \tag{1}$$

Note that $\widetilde{\mathrm{K}}\mathrm{s}(x)$ does not refer to CRNs in any direct way, so the tight asymptotic connection (Theorem 12) we establish may be surprising.

$\widetilde{\mathrm{K}}\mathrm{s}(x)$ is similar to the Kolmogorov complexity variant defined as $\mathrm{Ks}(x) = \min\{|p| + \log(\mathtt{space}(\mathcal{U}(p))) : \mathcal{U}(p, i) = x[i]\}$ by Allender, Kouckỳ, Ronneburger, and Roy [3] in that it additively mixes program size with the log of the space usage. There are two differences: (1) The program size component of $\widetilde{\mathrm{K}}\mathrm{s}$ is $|p|/\log|p|$ rather than $|p|$. The intuition is that a single chemical reaction can encode more than one bit of information; thus, a Turing machine program p can be converted to a "CRN program" with a number of reactions that is asymptotically smaller than the number of bits of p. (2) $\mathrm{Ks}(x)$ is defined with respect to programs that, given index i as input, output $x[i]$, the i'th bit of x, while our $\widetilde{\mathrm{K}}\mathrm{s}(x)$ is defined with respect to programs that (taking no input) directly output all of x. Thus $\widetilde{\mathrm{K}}\mathrm{s}(x) \geq \log|x|$, since the Turing machine must at least store the output integer, while $\mathrm{Ks}(x)$

[2] For stable *function* computation, an ordered subset of input species $\{X_1, X_2, \ldots, X_n\} \subset \mathcal{S}$ is also included; however, stable integer computation would be something along the lines of $f(1) = x$, so a single copy of the leader species serves as the "input" here.

may be smaller in principle. Due to the ability of efficient universal Turing machines to simulate each other efficiently, $\widetilde{\mathrm{Ks}}$ (like Ks) is invariant within multiplicative constants to the choice of universal Turing machine $\mathcal{U}$, as long as $\mathcal{U}$ is space-efficient. Note that if $\widetilde{\mathrm{Ks}}$ were not robust to the choice of $\mathcal{U}$, it could hardly be a universal measure.

It is worth noting that unlike $\mathrm{K}(x)$, $\widetilde{\mathrm{Ks}}(x)$ is computable. To see this, one can enumerate all programs for universal Turing machine $\mathcal{U}$ and run them in order from smallest to largest, stopping on the first machine that outputs x. Since the space usage of $\mathcal{U}(p)$ is included in $\widetilde{\mathrm{Ks}}$, we can terminate executions as soon as they start using too much space. This ensures that no execution will run forever, and so we are guaranteed to find the smallest p that outputs x.

2.3 Overview

Here, we give a high level overview for the constructions and results presented in the subsequent sections of this paper.

Our constructions rely on the ability of CRNs to "efficiently" simulate space-bounded Turing machines (in terms of program size and space usage, not time) by "efficiently" simulating bounded-count register machines. Section 3 details how to use a combination of previous results to achieve this. The first half of the section describes how to construct a CRN to faithfully simulate a bounded-count register machine. The second half of the section shows how to generate a large register machine bound (2^{2^n}) with very few species/reactions (n). While the latter result is from previous work [11], we translate their construction from a commutative semigroup presentation into a chemical reaction network.

In Section 4, we present a method for constructing a CRN $\mathcal{C}_x$ which (optimally) haltingly computes n-bit integer x with $|\mathcal{C}_x| = \mathcal{O}(n/\log n)$ by using a permutation code (Theorem 6). The idea of the construction is to generate a specified permutation and convert that permutation to a mapped target integer x. This construction relies on the "efficient" bounded-count register machine and space-bounded Turing machine simulations.

We then show how to use our permutation construction to achieve an optimal encoding (within global multiplicative constants) for algorithmically compressible integers in Section 5. Here, we use our permutation code technique to "unpack" a Turing machine program that that outputs x, resulting in a CRN that haltingly computes x with $\mathcal{O}(\widetilde{\mathrm{Ks}})$ reactions (Theorem 7). Afterwards, we use a result from Künnemann et al. [24] to show that the size of our constructed CRN is within multiplicative constants of the optimal size of a CRN that stably computes x, denoted $\mathrm{K}_{\mathrm{crn}}(x)$ (Lemma 11). The results of the paper culminate with us connecting $\mathrm{K}_{\mathrm{crn}}(x)$ and $\widetilde{\mathrm{Ks}}$ in Theorem 12 (our main theorem), which is directly implied by the combination of Theorem 7 and Lemma 11.

Lastly, we present some open questions for future work in Section 6.

3 Efficient Simulation of Bounded Register Machines

3.1 Register machines

A register machine is a finite state machine along with a fixed number of registers, each with non-negative integer counts. The two fundamental instructions for a register machine are increment $inc(r_i, s_j)$ and decrement $dec(r_i, s_j, s_k)$. The first instruction increments register r_i and transitions the machine to state s_j. The second instruction decrements register r_i if it is non-zero and transitions the machine to state s_j, otherwise the machine just transitions to state s_k. We also consider the more advanced instruction of $copy(r_i, r_j, s_k)$, which adds the value of register r_i to register r_j, i.e., it is equivalent to the assignment statement $r_j := r_j + r_i$

(note that the value is preserved in r_i). It is clear that *copy* can be constructed with a constant number of register machine states. In fact, register machines are known to be Turing-universal with three registers [30].[3]

In [36], a simple CRN construction was shown to simulate register machines with some possibility of error (thus not directly compatible with stable computation). The source of the error is due to the zero-checking in a *dec* instruction. For the simulation, the CRN has a finite set of species (one for each register and one for each state of the register machine) and a finite set of reactions (one for each instruction in the register machine program). Each $inc(r_i, s_j)$ instruction corresponds to the reaction $S_{j'} \to R_i + S_j$, and each $dec(r_i, s_j, s_k)$ instruction to two reactions $S_{j'} + R_i \to S_j$ and $S_{j'} \to S_k$. In the chemical reaction network implementation of a *dec* instruction, the two reactions are competing for the state species $S_{j'}$. While in general this is an unavoidable problem, in the special case that the maximum value in our counters is bounded by a constant, we can remedy this following the idea from [28] as follows.

Let's consider bounded registers that can contain a value no greater than $b \in \mathbb{N}$. For each register r_i, we can use two species R_i^A and R_i^I as "active" and "inactive" species for register r_i, respectively. The idea is that the total sum of the counts of species R_i^A and R_i^I is always equal to b: whenever one is consumed, the other is produced. Now, an $inc(r_i, s_j)$ instruction could be implemented with the reaction $S_{j'} + R_i^I \to R_i^A + S_j$, and a $dec(r_i, s_j, s_k)$ instruction could be implemented with the reactions $S_{j'} + R_i^A \to R_i^I + S_j$ and $S_{j'} + bR_i^I \to bR_i^I + S_k$. With this approach, register r_i has a zero count exactly when inactive species R_i^I has a count of b, and so we can zero-check without error. Notice that this approach uses reactions with a large stoichiometric coefficient b. At this point, there are two issues to be addressed: (1) how to generate an initial b count of inactive species R_i^I, and (2) how to transform the reactions into a series of third-order reactions (avoiding the large stoichiometric coefficient b).

Let's first consider a very simple construction which addresses the above concerns, albeit suboptimally. Suppose $b = 2^n$ is a power of two. To handle (1), we can initially produce count b of R_i^I from a single copy of A_1 using $O(\log b)$ species with reactions

$$\begin{aligned} A_1 &\to 2A_2 \\ A_2 &\to 2A_3 \\ &\vdots \\ A_n &\to R_i^I. \end{aligned}$$

To handle (2), we can transform the decrement reactions into a series of n bimolecular reactions by adding reversible versions of the reactions from (1) and "counting down" to some unique zero count indicator species C_1:

$$\begin{aligned} R_i^I &\rightleftarrows C_n \\ 2C_n &\rightleftarrows C_{n-1} \\ &\vdots \\ 2C_2 &\rightleftarrows C_1. \end{aligned}$$

Then C_1 is producible if and only if R_i^I had count $\geq b$, so the reaction $S_j + C_1 \to S_k + C_1$ implements the "jump to state k if $r_i = 0$" portion of the $dec(r_i, s_j, s_k)$ command. This

[3] Turing-universality has also been shown for machines with two registers, but only when a nontrivial encoding of the input/output is allowed [30, 33].

construction allows an error-free simulation of a register machine with counters with bound b exponential in the number of species. Now we discuss a more sophisticated construction, based on previous results [11, 28], that achieves a counter bound b that is *doubly* exponential in the number of species.

3.2 Counting to 2^{2^n} with n species

The CRN constructions in this paper simulate bounded register machines in the manner discussed previously. Since we are focused on reducing the size of our CRN, we want to do this simulation with as few species (reactions) as possible. Fortunately, we can rely on established results from prior work to do this. Lipton provided a construction for which the largest producible amount of a species is a doubly exponential count [28]. However, this amount is only produced non-deterministically and (most) paths produce less. Cardoza et al. went on to present a fully reversible system that can achieve this doubly exponential count as well [11]. Further, their system is halting in the sense that a new species is produced precisely when the maximum amount is reached.

While Cardoza et al. [11] describe their construction in the language of commutative semigroup presentations, we present a modified construction in Figure 1 articulated as a CRN. In the figure and in the text below, we use the "box" notation to indicate *meta-reactions*, which correspond to a set of reactions. Note that in Lemma 4 we will see that the combined behavior of the reactions in a meta-reaction module faithfully implement the meta-reaction semantics. By construction, the sets of reactions that meta-reactions expand to overlap, and we include only one copy of any repeated reaction. Each layer of the construction introduces $\mathcal{O}(1)$ more reactions and species – 9 reactions ((1)–(9)) and 9 species (S_i^k, H_i^k, X_i^k, $T1_i^k$, $T2_i^k$, $C1_i^k$, $C2_i^k$, $C3_i^k$, $C4_i^k$) for each $i \in \{1, 2, 3, 4\}$.

The idea of the construction is to produce (or consume) a doubly exponential count of species X by recursively producing (or consuming) quadratically more X's than the previous layer. Each species type performs a different role. X is the counting species to be generated or consumed. S starts the process to generate/consume many molecules of species X. T transforms different types of X species into one another. H indicates that the generation/consumption process has completed. C "cleans up" the H species. Reaction (5) in the meta-reaction implementation (which converts X_2^{k-1} into X_3^{k-1}) changes based on i. If $i \in \{1, 2\}$, then X_i appears as a product and is generated by this reaction. If $i \in \{3, 4\}$, the X_i appears as a reactant and is consumed by this reaction. A high level diagram of a layer-k meta-reaction is shown in Figure 2, which is helpful in understanding the behavior of the system.

▶ **Definition 1.** *Let* $\mathbf{c}$ *be a configuration of CRN* $\mathcal{C}$ *given above. We say* $\mathbf{c}$ *is* well-led *if* $\mathbf{c}(S_*^*) + \mathbf{c}(H_*^*) + \mathbf{c}(T_*^*) = 1$ *where the notation* S_*^* *denotes any species with label* S*, regardless of the subscript or superscript. In other words, there is only a single leader in the system and it either has the label* S*,* H*, or* T*. We call species* S_*^**,* H_*^**, and* T_*^* leader species*.*

▶ **Observation 2.** *Every reaction has exactly one leader species as a reactant, and exactly one leader species as a product.*

The following is immediate from Observation 2:

▶ **Corollary 3.** *Let* $\mathbf{c}$ *be a well-led configuration of CRN* $\mathcal{C}$ *given above. Then any configuration* $\mathbf{d}$ *such that* $\mathbf{c} \rightsquigarrow \mathbf{d}$ *is also well-led. In other words, the well-led property is forward invariant.*

Informally, the observation above together with the well-led condition implies that we can reason about the meta-reactions in isolation, without fear of cross-talk – because while one meta-reaction is executing, no reactions outside of it are applicable. This allows us to inductively prove the main result of this section:

Base reactions:

$$S_1^0 \rightleftarrows H_1^0 + 2X_1^0$$
$$S_2^0 \rightleftarrows H_2^0 + 2X_2^0$$
$$2X_3^0 + S_3^0 \rightleftarrows H_3^0$$
$$2X_4^0 + S_4^0 \rightleftarrows H_4^0$$

Layer-k meta-reactions:

$$\boxed{S_1^k \rightleftharpoons H_1^k + 2^{2^k} X_1^k}$$
$$\boxed{S_2^k \rightleftharpoons H_2^k + 2^{2^k} X_2^k}$$
$$\boxed{2^{2^k} X_3^k + S_3^k \rightleftharpoons H_3^k}$$
$$\boxed{2^{2^k} X_4^k + S_4^k \rightleftharpoons H_4^k}$$

Implementation of layer-k meta-reaction:

$$S_i^k \rightleftarrows C1_i^k + S_1^{k-1} \quad (1)$$
$$\boxed{S_1^{k-1} \rightleftharpoons H_1^{k-1} + 2^{2^{k-1}} X_1^{k-1}}$$
$$C1_i^k + H_1^{k-1} \rightleftarrows T1_i^k \quad (2)$$
$$T1_i^k + X_1^{k-1} \rightleftarrows C2_i^k + X_4^{k-1} + S_2^{k-1} \quad (3)$$
$$\boxed{S_2^{k-1} \rightleftharpoons H_2^{k-1} + 2^{2^{k-1}} X_2^{k-1}}$$
$$C2_i^k + H_2^{k-1} \rightleftarrows T2_i^k \quad (4)$$
$$X_i^k + T2_i^k + X_2^{k-1} \rightleftarrows T2_i^k + X_3^{k-1} + X_i^k \quad (5)$$
$$T2_i^k \rightleftarrows C3_i^k + S_3^{k-1} \quad (6)$$
$$\boxed{2^{2^{k-1}} X_3^{k-1} + S_3^{k-1} \rightleftharpoons H_3^{k-1}}$$
$$C3_i^k + H_3^{k-1} \rightleftarrows T1_i^k \quad (7)$$
$$C3_i^k + H_3^{k-1} \rightleftarrows C4_i^k + S_4^{k-1} \quad (8)$$
$$\boxed{2^{2^{k-1}} X_4^{k-1} + S_4^{k-1} \rightleftharpoons H_4^{k-1}}$$
$$C4_i^k + H_4^{k-1} \rightleftarrows H_i^k \quad (9)$$

Figure 1 Doubly exponential counting construction. (Left) The base reactions and layer-k meta-reactions. We use the box notation to indicate meta-reactions, which correspond to a set of reactions. Note that the reactions corresponding to the different meta-reactions overlap; when all the meta-reactions are expanded we include only one reaction copy. (Right) Explicit reactions for the layer-k meta-reaction in terms of reactions and other meta-reactions. The core functionality is the same for any i, but reaction (5) either generates X_i^k's (if $i \in \{1,2\}$) or consumes X_i^k's (if $i \in \{3,4\}$).

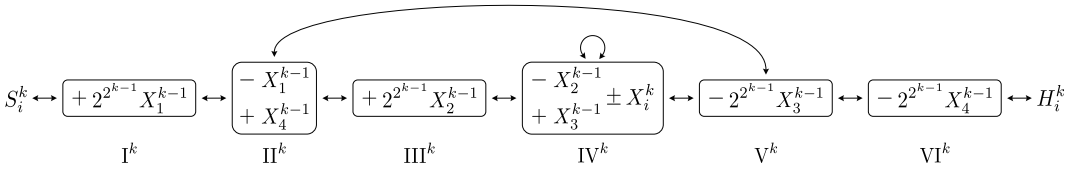


Figure 2 A visualization of the states for a layer-k meta-reaction. The state transitions effectively execute a nested loop. In order to iterate the outer loop (transition from V $\to$ II), the inner loop IV must be executed $2^{2^{k-1}}$ times. And in order to leave state VI and produce an H, the outer loop must be executed $2^{2^{k-1}}$. So, state IV must be executed a total of 2^{2^k} times, which either produces or consumes that many X_i^k's depending on the type of meta-reaction.

▶ **Lemma 4** (Production). *Consider the CRN implementing* $\boxed{S_i^k \rightleftharpoons H_i^k + 2^{2^k} X_i^k}$. *For any* $n \in \mathbb{N}$, *let* $\mathbf{s} = \{nX_i^k, 1S_i^k\}$ *and* $\mathbf{h} = \{(2^{2^k} + n)X_i^k, 1H_i^k\}$, *and let* $\mathbf{c}$ *be any configuration reachable from* $\mathbf{s}$ *or* $\mathbf{h}$. *Then: (a) Both* $\mathbf{s}$ *and* $\mathbf{h}$ *are reachable from* $\mathbf{c}$. *(b) If* $\mathbf{c}$ *contains* S_i^k *then* $\mathbf{c} = \mathbf{s}$, *and if* $\mathbf{c}$ *contains* H_i^k *then* $\mathbf{c} = \mathbf{h}$.

▶ **Lemma 5** (Consumption). *Consider the CRN implementing* $\boxed{2^{2^k} X_i^k + S_i^k \rightleftharpoons H_i^k}$. *For any* $n \in \mathbb{N}$, *let* $\mathbf{s} = \{(2^{2^k} + n)X_i^k, 1S_i^k\}$ *and* $\mathbf{h} = \{nX_i^k, 1H_i^k\}$, *and let* $\mathbf{c}$ *be any configuration reachable from* $\mathbf{s}$ *or* $\mathbf{h}$. *Then: (a) Both* $\mathbf{s}$ *and* $\mathbf{h}$ *are reachable from* $\mathbf{c}$. *(b) If* $\mathbf{c}$ *contains* S_i^k *then* $\mathbf{c} = \mathbf{s}$, *and if* $\mathbf{c}$ *contains* H_i^k *then* $\mathbf{c} = \mathbf{h}$.

Proof. (Of Lemma 4 and Lemma 5, Sketch) Both lemmas are proven by induction over the layers of the construction. The base case ($k = 1$) can be checked by inspection. Now assume the lemmas are true for $k - 1$ layers, and we want to prove them true for k layers.

First we argue that the construction is correct if the $k - 1$ layer meta-reactions are "atomic" and occur in one step. As visualized in Figure 2, the CRN iterates through a nested loop process. Each state transition (states I^k through VI^k) is coupled to a conversion of the leader species; the well-led condition ensures that the CRN is in exactly one state at any given time. Each net forward traversal of the outer loop converts a X_1^{k-1} to X_4^{k-1}, and each forward traversal of the inner loop converts a X_2^{k-1} to X_3^{k-1}. Step I^k makes $2^{2^{k-1}} X_1^{k-1}$, bounding the net maximum number of times that the outer loop can happen in the forward direction. Step III^k makes $2^{2^{k-1}} X_2^{k-1}$, bounding the net maximum number of times that the inner loop can happen in the forward direction for every net forward traversal of the outer loop. This implies that reaction (5) can fire at most a net total 2^{2^k} times (producing at most a net total $2^{2^k} X_i^k$'s).

Step V^k consumes $2^{2^{k-1}} X_3^{k-1}$, requiring the net total number of forward traversals of the inner loop to be at least $2^{2^{k-1}}$ for every net forward traversal of the outer loop. Step VI^k consumes $2^{2^{k-1}} X_4^{k-1}$, requiring the net total number of forward traversals of the outer loop to be at least $2^{2^{k-1}}$. This implies that reaction (5) must fire at least a net total 2^{2^k} times (producing at least a net total $2^{2^k} X_i^k$'s).

Thus, reaction (5) must be executed exactly 2^{2^k} times (producing exactly $2^{2^k} X_i^k$'s). Notice that an excess of X_i^k (as allowed by the statement of the lemma) does not affect the net total number of times reaction (5) can fire (forward or backward) since X_2^{k-1} and X_3^{k-1} are the limiting factors.

Now we need to make sure that this behavior is preserved once the meta-reactions are expanded to their constituent reactions. Each meta-reaction i in Figure 1 expands to some set R_i of reactions. First we note that for each meta-reaction, R_i overlaps with reactions not in R_i only over species S_i^{k-1}, H_i^{k-1}, and X_i^{k-1}. We are not worried about cross-talk in species S_i^{k-1} and H_i^{k-1} because of the well-led property. We may still be concerned, however, that external consumption of X_i^{k-1} might somehow interfere with the meta-reaction. Luckily, the well-led property and Observation 2 enforce that unless we have S_i^{k-1} or H_i^{k-1} (i.e., we are at the beginning or end of the meta-reaction), it is never the case that a reaction in R_i and a reaction not in R_i are applicable at the same time. Thus nothing outside the meta-reaction can change X_i^{k-1} while the meta-reaction is executing. ◀

Note that although we chose to write Lemma 4 and Lemma 5 separately, we could have just one kind of meta-reaction (production or consumption) and obtain the other kind by running the meta-reaction backward switching the roles of S and H. We include the two different versions because it is conceptually easier to just think about the intended execution being in the forward direction.

4 Optimal Encoding

4.1 Encoding Information in CRNs

In this section we discuss the encoding of an integer in a chemical reaction network. In the same sense as Kolmogorov-optimal programs for Turing machines, we consider a similar measure of optimality for chemical reaction networks. In particular, we ask the question, "what is the smallest chemical reaction network that can produce a desired count of a particular chemical species?"

A simple construction shows that x copies of some species can be produced using $\mathcal{O}(\log x)$ reactions. The idea is to have a reaction for each bit b_i of the binary expansion of x, and produce a copy of your output species in each reaction where $b_i = 1$. More concretely, consider $\log x$ reactions of the form $X_i \to 2X_{i+1}$ and $X_i \to 2X_{i+1} + Y$. For each bit b_i in the binary expansion of x, use the first reaction if $b_i = 0$ and use the second reaction if $b_i = 1$. Each species X_i will have a count equal to 2^i, and species Y will have a count equal to the sum of the powers of two that were chosen (which is x). While this simple construction generates x with $\log x$ reactions, it is not immediately clear how to improve upon it.

Our first result shows how to construct a CRN that can generate x copies of an output species (from an initial configuration with only a single molecule) yet uses only $\mathcal{O}(\log x / \log\log x)$ many reactions. This matches the lower bound dictated by Kolmogorov complexity (see end of Section 4), which suggests that the full power of CRNs is really being used in our construction. Our construction is achieved through the simulation of (space-bounded) Turing machines via the simulation of (space-bounded) register machines. A key aspect in this process is the ability of CRNs to use the previously discussed recursive counting technique to count very high with very few species (counting to 2^{2^k} with k species).

4.2 Our Construction

Now, we present an encoding scheme to produce count x of a particular species with $\mathcal{O}(n/\log n)$ CRN reactions, where $n = \log x$. In the simple CRN given in Section 4.1, each reaction encodes a single bit of x. In the optimized construction with k reactions, each reaction will encode $\log k$ bits instead.[4] A sketch of our construction is as follows:

Sketch: We start with a CRN in configuration $\mathbf{c}_1 = \{1L\}$ *and create a configuration* $\mathbf{c}_2 = \{1S_i, m_1R_1, m_2R_2, \ldots, m_kR_k\}$ *that represents a particular permutation of* k *distinct elements. We encode this permutation in the count of a species* I*, transforming configuration* $\mathbf{c}_2$ *into a configuration* $\mathbf{c}_3 = \{1S_j, mI\}$*. The count of species* I *can be interpreted as the input to a Turing machine, so we simulate a Turing machine that maps the permutation to a unique integer via Lehmer code/factorial number system [25, 34] (by choosing the right value of* k*, we can ensure there are sufficiently many permutations to let us map to* x*). This Turing machine simulation transforms configuration* $\mathbf{c}_3$ *into configuration* $\mathbf{c}_4 = \{1H, xY\}$*.*

▶ **Theorem 6.** *For any* $n \in \mathbb{N}$ *and any* n*-bit integer* x*, there exists a chemical reaction network* $\mathcal{C}_x$ *that haltingly computes* x *from initial configuration* $\{1L\}$ *with* $|\mathcal{C}_x| = \mathcal{O}(n/\log n)$*.*

Proof. First, we describe how to construct CRN $\mathcal{C}_x$ that haltingly computes x from starting configuration $\{1L\}$, then we describe the size of $|\mathcal{C}_x|$. Let $k = \lceil n/\log n \rceil$. We will map a permutation of k distinct elements to the integer x, and this value of k ensures there are at least x permutations. We break the construction into three primary steps.

[4] Adleman et al. [1] provided a clever base conversion trick for tile assembly programs. Here, we employ a permutation encoding trick to yield the same effect.

Step 1: $\{1L\} \rightsquigarrow \{1S_i, m_1R_1, m_2R_2, \ldots, m_kR_k\}$. We can transform $\{1L\}$ into a configuration $\{1S_i, m_1R_1, m_2R_2, \ldots, m_kR_k\}$ where $(m_1, m_2, \ldots, m_n)$ is a permutation of the integers 1 through k. This can be achieved with k registers and $2k$ register machine states. For example, to set the permutation $(2, 4, 3, 1)$, use instructions

$s_0 : inc(r_2, s_1)$
$s_1 : copy(r_2, r_4, s_2)$
$s_2 : inc(r_2, s_3)$
$s_3 : copy(r_2, r_1, s_4)$
$s_4 : inc(r_2, s_5)$
$s_5 : copy(r_2, r_3, s_6)$
$s_6 : inc(r_2, s_7)$

Step 2: $\{1S_i, m_1R_1, m_2R_2, \ldots, m_kR_k\} \rightsquigarrow \{1S_j, mI\}$. Now, we can transform configuration $\{1S_i, m_1R_1, m_2R_2, \ldots, m_kR_k\}$ into configuration $\{1S_j, mI\}$, encoding the permutation as the integer count m of species I. For each register r_i for i from 1 to k in order, we can decrement the register to 0. On each decrement, we double the count of I and then add 1 to it, i.e., appending a 1 to m's binary expansion. After the register reaches 0, before moving to the next register, we double the count of I again, appending a 0 to m's binary expansion. For example, if the permutation configuration was $\{3R_1, 1R_2, 2R_3\}$, the resulting count of I in binary would be

$$\underbrace{111}_{3}0\underbrace{1}_{1}0\underbrace{11}_{2}.$$

Step 3: $\{1S_j, mI\} \rightsquigarrow \{1H, xY\}$. At this point, we can consider the value in register I, expressed as a binary string, to be the input tape content for a Turing machine that maps the permutation to the integer x using a standard Lehmer code/factorial number system technique [25, 34]. The output of the Turing machine will be the count of Y in configuration $\mathbf{c}$ at the end of the computation (with $\mathbf{c}(Y) = x$). Our register machine will have a state species that corresponds to the halted state of the Turing machine – and such a species serves as our halting species H.

Now we argue the size of CRN $|\mathcal{C}_x| = \mathcal{O}(n/\log n)$, i.e., it uses $\mathcal{O}(k)$ reactions. The register machine program from Step 1 generates the permutation using k registers and $2k$ register machine states, which results in $\mathcal{O}(k)$ CRN reactions. The register machine program from Step 2 encodes the permutation as a binary number in register I using $\mathcal{O}(k)$ registers and $\mathcal{O}(k)$ register machine states, which also results in $\mathcal{O}(k)$ CRN reactions. Even a naive algorithm for the Turing machine from Step 3 maps the permutation to an integer using $\mathcal{O}(k^2 \log k)$ space ($\mathcal{O}(k^2)$ bits to store the initial permutation, $\mathcal{O}(k \log k)$ bits to store the Lehmer code, $\mathcal{O}(k^2 \log k)$ bits to store factorial bases 1! through k!, and $\mathcal{O}(k \log k)$ bits to store the integer x). Recall, a Turing machine using space $\mathcal{O}(k^2 \log k)$ can be simulated by a register machine with count bound $\mathcal{O}(2^{k^2 \log k})$ on its registers. This can in turn be simulated by a CRN via the construction of Section 3.2 with $\mathcal{O}(\log \log 2^{k^2 \log k}) = \mathcal{O}(\log k)$ reactions. Thus $\mathcal{O}(k)$ reactions suffices to simulate the register machine instructions as well as the bounded counters for our register machine to simulate this Turing machine. ◀

The above construction is optimal for almost all integers x in the following sense. Any CRN of $|\mathcal{C}|$ reactions, each with $\mathcal{O}(1)$ reactants and products, can be encoded in a string of length $\mathcal{O}(|\mathcal{C}| \log |\mathcal{C}|)$. Given an encoded CRN stably computing an integer x, a fixed-size

program can simulate it and return x. Thus $\mathrm{K}(x) \leq \mathcal{O}(|\mathcal{C}| \log |\mathcal{C}|)$. The pigeonhole principle argument for Kolmogorov complexity implies that $\mathrm{K}(x) < \lceil \log x \rceil - \Delta$ for at most $(1/2)^{\Delta}$ of all x [27]. Together these observations imply that there is a c such that for most x there does not exist a CRN $\mathcal{C}$ of size smaller than $cn/\log n$ that stably computes x.

5 Algorithmic Compression

The construction in Section 4 is optimal for incompressible integers (integers x where $\mathrm{K}(x) \approx |x|$, which is the case for "most" integers). Now we extend the construction to be optimal within global multiplicative constants for *all* integers. For algorithmically compressible integers x, there exists a p such that $\mathcal{U}(p) = x$ and $|p| < |x|$. We discuss the construction in Section 5.1 and we argue optimality of our construction in Section 5.2.

5.1 Our Construction

We now show how to fully exploit the encoding scheme and doubly exponential counter from Section 4 to achieve an optimal result for all integers. A sketch of our construction is as follows:

Sketch: Given a program p for a fixed Universal Turing Machine $\mathcal{U}$ such that $\mathcal{U}(p) = x$, we construct a CRN that simulates running p on $\mathcal{U}$ via a register machine simulation. The idea is to use $\mathcal{O}(|p|/\log|p|)$ reactions to encode p, and to use $\mathcal{O}(\log(\mathtt{space}(\mathcal{U}(p)))$ reactions for a counter machine simulation of $\mathcal{U}(p)$.

▶ **Theorem 7.** *For any integer x, there exists a CRN $\mathcal{C}_x$ that haltingly computes x from initial configuration $\{1L\}$ with $|\mathcal{C}_x| = \mathcal{O}(\widetilde{\mathrm{K}}\mathrm{s}(x))$.*

Proof. Let p be a program for a fixed Universal Turing Machine $\mathcal{U}$ such that $\mathcal{U}(p) = x$. We encode p in the manner provided by Theorem 6 using $\mathcal{O}(|p|/\log|p|)$ reactions. This results in p count of species Y (specifically, configuration $\{1H, pY\}$). Since haltingly-computing CRNs are composable via concatenation [12, 35], we can consider $\{1H, pY\}$ to be taken as the input for another system which simulates running $\mathcal{U}(p)$ via the previously described register machine method with bounded register count (Section 3.2). Again, we need enough species/reactions to ensure our bounded registers can count high enough. The registers must be able to store an integer that represents the current configuration of the Turing machine being simulated (at most this is $2^{\mathtt{space}(\mathcal{U}(p))}$). Since we have doubly exponential counters, an additional $\log\big(\mathtt{space}(\mathcal{U}(p))\big)$ species are needed to do this. So, the total size of our CRN is $\mathcal{O}\big(|p|/\log|p| + \log(\mathtt{space}(\mathcal{U}(p)))\big)$ and by choosing the program p that minimizes this expression, we see $|\mathcal{C}_x| = \mathcal{O}(\widetilde{\mathrm{K}}\mathrm{s}(x))$. ◀

It is interesting to note the appearance of our "space-aware" version of Kolmogorov complexity. Importantly, this notion is different from space-bounded Kolmogorov complexity that puts a limit on the space usage of the program that outputs x. This alternate version allows a trade-off between compact program descriptions and the space required to run those programs, which seems natural for systems like CRNs. Perhaps it is surprising that this (computable) measure of complexity shows up here, and at first it may seem like log of this space usage is a bit arbitrary, but we will show that this is indeed optimal (within global multiplicative constants) for CRNs.

5.2 Optimality

Here, we argue that size of CRN $\mathcal{C}_x$ from Theorem 7 is optimal. We begin by giving a definition for the size of the optimal CRN that haltingly computes an integer x.

▶ **Definition 8.** *For any integer x, define $\mathrm{K}_{\mathrm{crn}}(x) = \min\{|\mathcal{C}| : \text{CRN } \mathcal{C} \text{ haltingly computes } x\}$. In other words, $\mathrm{K}_{\mathrm{crn}}(x)$ is the size of the smallest CRN that haltingly computes x.*

Our argument relies on a Turing machine that solves the coverability problem for CRNs. We give the definition for this problem in Definition 9 and discuss its space complexity in Lemma 10.

▶ **Definition 9** (Coverability). *Given a CRN $\mathcal{C}$, initial configuration $\mathbf{s}$, and target configuration $\mathbf{u}$, does there exist a configuration $\mathbf{t} \geq \mathbf{u}$ such that $\mathbf{s} \rightsquigarrow \mathbf{t}$?*

Using the natural notion of problem size n for the specification of a coverability problem, Lipton provided a $2^{\Omega(\sqrt{n})}$ space lower bound for coverability [28], which was later improved to $2^{\Omega(n)}$ by Mayr and Meyer [29]. As for upper bounds, Rackoff provided an algorithm to decide coverability that uses $2^{\mathcal{O}(n \log n)}$ space [31]. Following this, Koppenhagen and Mayr gave an algorithm that decided coverability in $2^{\mathcal{O}(n)}$ space for *reversible* systems, closing the gap for this class of systems [23]. A recent result by Künnemann et al. also closes this gap [24] for Vector Addition Systems with States. Our work uses this latest result.

▶ **Lemma 10** (Implied by Theorem 3.3 from [24]). *Let CRN $\mathcal{C} = (\mathcal{S}, \mathcal{R})$ be a CRN that haltingly computes x. Then there exists an algorithm which solves coverability for CRN $\mathcal{C}$ for initial configuration $\{1L\}$ and target configuration $\{1H\}$ which uses $2^{\mathcal{O}(|\mathcal{C}|)}$ space.*

Proof. This result follows from Theorem 3.3 from the recent work by Künnemann et al. [24]. There, the authors consider the problem of coverability in Vector Addition Systems with States (VASS). They show that if the answer to coverability is yes, the length of the longest path is $n^{2^{\mathcal{O}(d)}}$ where n is the maximum value change of any transition and d is the dimension of the vector. For us, $n = 2$ since each reaction has at most two reactants/products and $d = |\mathcal{S}|$. While vector addition systems are not capable of "catalytic" transitions, it is known that the same effect can be achieved by decomposing transitions into two vector additions. So the path length would at most double for our systems.

With this bound on the path length, we can consider an algorithm that non-deterministically explores the state space of $\mathcal{C}$ (from starting configuration $\{1L\}$) by simulating reactions on a current configuration of the system until a configuration that covers $\{1H\}$ is found. By Savitch's Theorem [32], this can be converted to a deterministic algorithm using the same space: $\mathcal{O}(|\mathcal{R}| \log |\mathcal{R}|)$ bits to hold a description of $\mathcal{C}$, $2^{\mathcal{O}(|\mathcal{S}|)}$ bits for a path length counter, and $2^{\mathcal{O}(|\mathcal{S}|)} \cdot \log(|S|)$ bits to store the current configuration of $\mathcal{C}$. All of these values are absorbed under a $2^{\mathcal{O}(|\mathcal{C}|)}$ bound. ◀

With this space bound on the coverability problem established, we can now argue that the size of our constructed CRN from Theorem 7 is asymptotically equal to the size of the the smallest CRN that haltingly computes x.

▶ **Lemma 11.** *For all $x \in \mathbb{N}$, letting $\mathcal{C}_x$ be the CRN from Theorem 7, $|\mathcal{C}_x| = \Theta(\mathrm{K}_{\mathrm{crn}}(x))$.*

Proof. Clearly $|\mathcal{C}_x| = \Omega(\mathrm{K}_{\mathrm{crn}}(x))$, by definition of $\mathrm{K}_{\mathrm{crn}}(x)$ and since $\mathcal{C}_x$ from Section 5 is an instance of a CRN that haltingly computes x. Now, we argue that $|\mathcal{C}_x| = \mathcal{O}(\mathrm{K}_{\mathrm{crn}}(x))$. The big picture is that one of the programs over which $\widetilde{\mathrm{K}}\mathrm{s}(x)$ is minimized in the construction of $\mathcal{C}_x$ is the program solving coverability for the optimal CRN for generating x.

Start with the CRN $\mathcal{K} = (\mathcal{S}_\mathcal{K}, \mathcal{R}_\mathcal{K})$ that haltingly computes x with optimal size $|\mathcal{K}| = \mathrm{K}_{\mathrm{crn}}(x) = n$. Consider program $p_\mathcal{K}$ that solves coverability for $\mathcal{K}$ with initial configuration $\{1L\}$ and target configuration $\{1H\}$, and outputs x. Now, with that program $p_\mathcal{K}$, build a CRN $\mathcal{K}'$ by following our construction for Theorem 7. We know $|p_\mathcal{K}| = \mathcal{O}(n \log n)$ so our final CRN needs $\mathcal{O}(n)$ reactions to encode $p_\mathcal{K}$ (by Theorem 6). By Lemma 10, we know that the space usage of $\mathcal{U}(p_\mathcal{K})$ is $2^{\mathcal{O}(|\mathcal{K}|)}$, so our final CRN needs $\mathcal{O}(|\mathcal{K}|)$ additional reactions to have large enough registers for the simulation of $\mathcal{U}(p_\mathcal{K})$. Thus, the total size of our final CRN is $\mathcal{O}(|\mathcal{K}| + |\mathcal{K}|) = \mathcal{O}(\mathrm{K}_{\mathrm{crn}}(x))$. ◀

The following theorem, which is the main result of our paper, follows immediately from Theorem 7 and Lemma 11. It characterizes the optimal number of reactions haltingly computing a number x using the space-aware Kolmogorov complexity measure $\widetilde{\mathrm{K}}\mathrm{s}$ defined in Section 2.2.

▶ **Theorem 12.** *For all* $x \in \mathbb{N}$, $\mathrm{K}_{\mathrm{crn}}(x) = \Theta(\widetilde{\mathrm{K}}\mathrm{s}(x))$.

Although, as mentioned above, CRN stable computation is not Turing universal, the theorem underlines its essential connection to space-bounded Turing machine computation.

6 Open Questions

Our results rely on the fact that we consider CRNs that perform halting computation – the end of the computation is indicated by the production of a designated halting species. This constraint, intuitively that the systems know when they have finished a computation, is rather strong. It is known that a much larger class of functions can be stably computed than can be haltingly computed [15]. It remains an open question if lifting this halting requirement (and allowing just stable computation) reduces the reaction complexity.

It is also worth noting that our approach starts with exactly one copy of a special leader species. Recently, Czerner showed that leaderless protocols are capable of deciding doubly exponential thresholds [16]. While starting in some uniform state and converging to a specific state would be a better expression of "chemical self-organization," their construction seems incompatible with our register machine simulation. Leaderless stable integer computation remains an area for future work.

Making a tight connection between stable integer computation and counting predicate computation commonly studied in population protocols [16] also remains open. We can easily follow the halting generation of a specific amount of x by running the "less-than-or-equal-to" predicate, thereby converting our constructions to compute a counting predicate with only a constant more reactions. This gives a new general upper bound on the complexity of counting in terms of $\widetilde{\mathrm{K}}\mathrm{s}(x)$. However, it is unclear whether counting predicate constructions carry over to the generation problem, leaving it open whether counting may be easier.

Our notion of "space-aware" Kolmogorov complexity $\widetilde{\mathrm{K}}\mathrm{s}$ is interesting in its own right. While the similar quantity Ks has been previously studied in the context of computational complexity theory [3] (see also Section 2.2), it is not clear which properties proven of Ks carry over to $\widetilde{\mathrm{K}}\mathrm{s}$. Although the robustness to the choice of $\mathcal{U}$ carries over, other properties may not. For example, it is not obvious whether our results still hold if we consider programs that output a single bit of x at a time (like Ks does).

A core piece of this work is simulating space-bounded Turing machines, so it is very natural to extend the discussion to Boolean circuits (computing functions $\phi : \{0,1\}^n \to \{0,1\}$). When attempting to compute Boolean functions with CRNs, one may be tempted to directly implement a Boolean circuit by creating $\mathcal{O}(1)$ reactions per gate in the circuit. However,

our results imply that reaction complexity can be improved by doing a space-bounded Turing machine simulation instead – when the circuit is algorithmically "compressible." An important class of such compressible circuits are uniform circuits, i.e., those constructable by a fixed Turing machine given an input size. Prior work established a quadratically tight connection between the depth of uniform circuits and Turing machine space [9]. Further investigation into optimal Boolean function computation is warranted.

References

1 Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 740–748, 2001.

2 Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2221–2239. SIAM, 2018.

3 Eric Allender, Michal Kouckỳ, Detlef Ronneburger, and Sambuddha Roy. The pervasive reach of resource-bounded Kolmogorov complexity in computational complexity theory. *Journal of Computer and System Sciences*, 77(1):14–40, 2011.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 290–299, 2004.

5 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299, 2006.

6 Petra Berenbrink, George Giakkoupis, and Peter Kling. Optimal time and space leader election in population protocols. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 119–129, 2020.

7 Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct population protocols for presburger arithmetic. *arXiv preprint arXiv:1910.04600*, 2019.

8 Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: on the minimal size of population protocols. *arXiv preprint arXiv:1801.00742*, 2018.

9 Allan Borodin. On relating time and space to size and depth. *SIAM journal on computing*, 6(4):733–744, 1977.

10 Daniele Cappelletti, Andrés Ortiz-Muñoz, David F Anderson, and Erik Winfree. Stochastic chemical reaction networks for robustly approximating arbitrary probability distributions. *Theoretical Computer Science*, 801:64–95, 2020.

11 E Cardoza, R Lipton, and Albert R Meyer. Exponential space complete problems for Petri nets and commutative semigroups (preliminary report). In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 50–54, 1976.

12 Cameron Chalk, Niels Kornerup, Wyatt Reeves, and David Soloveichik. Composable rate-independent computation in continuous chemical reaction networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 18(1):250–260, 2019.

13 Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural computing*, 13:517–534, 2014.

14 Ben Chugg, Hooman Hashemi, and Anne Condon. Output-oblivious stochastic chemical reaction networks. In *22nd International Conference on Principles of Distributed Systems (OPODIS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

15 Rachel Cummings, David Doty, and David Soloveichik. Probability 1 computation with chemical reaction networks. *Natural Computing*, 15(2):245–261, 2016. Special issue of invited papers from DNA 2014. doi:10.1007/s11047-015-9501-x.

16 Philipp Czerner. Leaderless population protocols decide double-exponential thresholds. *arXiv preprint arXiv:2204.02115*, 2022.

17 Philipp Czerner and Javier Esparza. Lower bounds on the state complexity of population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 45–54, 2021.

18 Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. Fast and succinct population protocols for presburger arithmetic. *arXiv preprint arXiv:2202.11601*, 2022.

19 David Doty, Mahsa Eftekhari, Leszek Gąsieniec, Eric Severson, Przemyslaw Uznański, and Grzegorz Stachowiak. A time and space optimal stable population protocol solving exact majority. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1044–1055. IEEE, 2022.

20 Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets—a survey. *Journal of Information Processes and Cybernetics*, 3:143–160, 1994.

21 Daniel T Gillespie. Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, 58:35–55, 2007.

22 Richard M Karp and Raymond E Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969.

23 Ulla Koppenhagen and Ernst W Mayr. Optimal algorithms for the coverability, the subword, the containment, and the equivalence problems for commutative semigroups. *Information and Computation*, 158(2):98–124, 2000.

24 Marvin Künnemann, Filip Mazowiecki, Lia Schütze, Henry Sinclair-Banks, and Karol Węgrzycki. Coverability in VASS revisited: Improving Rackoff's bound to obtain conditional optimality. *arXiv preprint arXiv:2305.01581*, 2023.

25 Derrick H Lehmer. Teaching combinatorial tricks to a computer. In *Proc. Sympos. Appl. Math. Combinatorial Analysis*, volume 10, pages 179–193, 1960.

26 Jérôme Leroux. State complexity of protocols with leaders. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 257–264, 2022.

27 Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

28 Richard Lipton. The reachability problem requires exponential space. *Research Report 62. Department of Computer Science, Yale University*, 1976.

29 Ernst W Mayr and Albert R Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in mathematics*, 46(3):305–329, 1982.

30 Marvin Lee Minsky. *Computation*. Prentice-Hall Englewood Cliffs, 1967.

31 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

32 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

33 Rich Schroeppel. A two counter machine cannot calculate 2^N. Technical report, Massachusetts Institute Of Technology, Artificial Intelligence Lab, 1972.

34 Robert Sedgewick. Permutation generation methods. *ACM Computing Surveys (CSUR)*, 9(2):137–164, 1977.

35 Eric E Severson, David Haley, and David Doty. Composable computation in discrete chemical reaction networks. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 14–23, 2019.

36 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *natural computing*, 7:615–633, 2008.

Complexity of Reconfiguration in Surface Chemical Reaction Networks

Robert M. Alaniz
University of Texas Rio Grande Valley, Edinburg, TX, USA

Josh Brunner
Massachusetts Institute of Technology, Cambridge, MA, USA

Michael Coulombe
Massachusetts Institute of Technology, Cambridge, MA, USA

Erik D. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA

Jenny Diomidova
Massachusetts Institute of Technology, Cambridge, MA, USA

Timothy Gomez
Massachusetts Institute of Technology, Cambridge, MA, USA

Elise Grizzell
Massachusetts Institute of Technology, Cambridge, MA, USA

Ryan Knobel
Massachusetts Institute of Technology, Cambridge, MA, USA

Jayson Lynch
Massachusetts Institute of Technology, Cambridge, MA, USA

Andrew Rodriguez
University of Texas Rio Grande Valley, Edinburg, TX, USA

Robert Schweller
University of Texas Rio Grande Valley, Edinburg, TX, USA

Tim Wylie
University of Texas Rio Grande Valley, Edinburg, TX, USA

Abstract

We analyze the computational complexity of basic reconfiguration problems for the recently introduced surface Chemical Reaction Networks (sCRNs), where ordered pairs of adjacent species nondeterministically transform into a different ordered pair of species according to a predefined set of allowed transition rules (chemical reactions). In particular, two questions that are fundamental to the simulation of sCRNs are whether a given configuration of molecules can ever transform into another given configuration, and whether a given cell can ever contain a given species, given a set of transition rules. We show that these problems can be solved in polynomial time, are NP-complete, or are PSPACE-complete in a variety of different settings, including when adjacent species just swap instead of arbitrary transformation (swap sCRNs), and when cells can change species a limited number of times (k-burnout). Most problems turn out to be at least NP-hard except with very few distinct species (2 or 3).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases Chemical Reaction Networks, reconfiguration, hardness

Digital Object Identifier 10.4230/LIPIcs.DNA.29.10

Related Version *Full Version*: `https://arxiv.org/abs/2303.15556`

1 Introduction

The ability to engineer molecules to perform complex tasks is an essential goal of molecular programming. A popular theoretical model for investigating molecular systems and distributed systems is Chemical Reaction Networks (CRNs) [6, 26]. The model abstracts chemical reactions to independent rule-based interactions that creates a mathematical framework

equivalent [8] to other well-studied models such as Vector Addition Systems [18] and Petri nets [24]. CRNs are also interesting for experimental molecular programmers, as examples have been built using DNA strand displacement (DSD) [27].

Abstract Surface Chemical Reaction Networks (sCRNs) were introduced in [25] as a way to model chemical reactions that take place on a surface, where the geometry of the surface is used to assist with computation. In this work, the authors gave a possible implementation of the model similar to ideas of spatially organized DNA circuits [21]. This strategy involves DNA strands being anchored to a DNA origami surface. These strands allow for "species" to be attached. Fuel complexes are pumped into the system, which perform the reactions. While these reactions are more complex than what has been implemented in current lab work, it shows a route to building these types of networks.

1.1 Motivation

Feed-Forward circuits using DNA hairpins anchored to a DNA origami surface were implemented in [5]. This experiment used a single type of fuel strand. The copies of the fuel strand attached to the hairpins and were able to drive forward the computation.

A similar model was proposed in [9], which modeled DNA walkers moving along tracks. These tracks have guards that can be opened or closed at the start of computation by including or omitting specific DNA species at the start. DNA walkers have provided interesting implementations such as robots that sort cargo on a surface [29].

A new variant of surface CRNs we introduce is the k-burnout model in which cells can switch states at most k time before being stuck in their final state. This models the practical scenario in which state changes expend some form of limited fuel to induce the state change. Specific experimental examples of this type of limitation can be seen when species encode "fire-once" DNA strand replacement reactions on the surface of DNA origami, as is done within the Signal Passing Tile Model [22].

1.2 Previous Work

The initial paper on sCRNs [25] gave a 1D reversible Turing machine as an example of the computational power of the model. They also provided other interesting constructions such as building dynamic patterns, simulating continuously active Boolean logic circuits, and cellular automata. Later work in [7] gave a simulator of the model, improved some results of [25], and gave many open problems- some of which we answer here.

In [2], the authors introduce the concept of swap reactions. These are reversible reactions that only "swap" the positions of the two species. The authors of [2] gave a way to build feed-forward circuits using only a constant number of species and reactions. These swap reactions may have a simpler implementation and also have the advantage of the reverse reaction being the same as the forward reaction, which makes it possible to reuse fuel species.

A similar idea for swap reactions on a surface that has been studied theoretically are friends-and-strangers graphs [10]. This model was originally introduced to generalize problems such as the 15 Puzzle and Token Swapping. In the model, there is a location graph containing uniquely labeled tokens and a friends graph with a vertex for every token, and an edge if they are allowed to swap locations when adjacent in the location graph. The token swapping problem can be represented with a complete friends graph, and the 15 puzzle has a grid graph as the location graph and a star as the friends graph (the 'empty square' can swap with any other square). Swap sCRNs can be described as multiplicities friends-and-strangers graph [19], which relax the unique restriction, with the surface grid (in our case the square grid) as the location graph and the allowed reactions forming the edges of the friends graph.

Table 1 Summary of our and known complexity results for sCRN reconfiguration problems, depending on the type of sCRN, number of species, and number of rules. All problems are contained in PSPACE, while all k-burnout problems are in NP.

Problem	Type	Graph	Species	Rules	Result	Ref
Reconfiguration	sCRN	1D	17	67	PSPACE-complete	[25]
1-Reconfiguration	Swap sCRN	Grid	4	3	PSPACE-complete	Thm. 3
1-Reconfiguration	Swap sCRN	Any	≤ 3	Any	P	Thm. 6
1-Reconfiguration	Swap sCRN	Any	Any	≤ 2	P	Thm. 6
Reconfiguration	Swap sCRN	Grid	4	3	PSPACE-complete	Thm. 4
Reconfiguration	Swap sCRN	Any	≤ 3	Any	P	Thm. 5
Reconfiguration	Swap sCRN	Any	Any	≤ 2	P	Thm. 5
Reconfiguration	2-burnout	Grid	3	1	NP-complete	Thm. 7
1-Reconfiguration	1-burnout	Grid	17	40	NP-complete	Thm. 8
Reconfiguration	sCRN	Grid	≥ 3	1	NP-complete	Cor. 15
Reconfiguration	sCRN	Any	≤ 2	1	P	Thm. 11

1.3 Our Contributions

In this work, we focus on two main problems related to sCRNs. The first is the reconfiguration problem, which asks given two configurations and a set of reactions, can the first configuration be transformed to the second using the set of reactions. The second is the 1-reconfiguration problem, which asks whether a given cell can ever contain a given species. Our results are summarized in Table 1. The first row of the table comes from the Turing machine simulation in [25] although it is not explicitly stated. The size comes from the smallest known universal reversible Turing machine [20] (see [30] for a survey on small universal Turing machines.)

We first investigate swap reactions in Section 3. We prove both problems are PSPACE-complete using only four species and three swap reactions. For reconfiguration, we show this complexity is tight by showing with three or less species and only swap reactions the problem is in P.

In Section 4, we study a restriction on surface CRNs called k-burnout where each species is guaranteed to only transition k times. This is similar to the freezing restriction from Cellular Automata [14, 15, 28] and Tile Automata [4]. We start with a simple reduction showing reconfiguration is NP-complete in 2-burnout. This is also of interest since the reduction only uses three species types and a reaction set of size one. For 1-reconfiguration, we show the problem is also NP-complete in 1-burnout sCRNs. This reduction uses a constant number of species.

In Section 5, we analyze reconfiguration for all sCRNs that have a reaction set of size one. For the case of only two species, we show for every possible reaction, the problem is solvable in polynomial time. With three species or greater, we show that reconfiguration is NP-complete. The hardness comes from the reduction in burnout sCRNs.

Finally, in Section 6, we conclude the paper by discussing the results as well as many open questions and other possible directions for future research related to surface CRNs.

2 Surface CRN model

Chemical Reaction Network. A *chemical reaction network (CRN)* is a pair $\Gamma = (S, R)$ where S is a set of species and R is a set of reactions, each of the form $A_1 + \cdots + A_j \to B_1 + \cdots + B_k$ where $A_i, B_i \in S$. (We do not define the dynamics of general CRNs, as we do not need them here.)

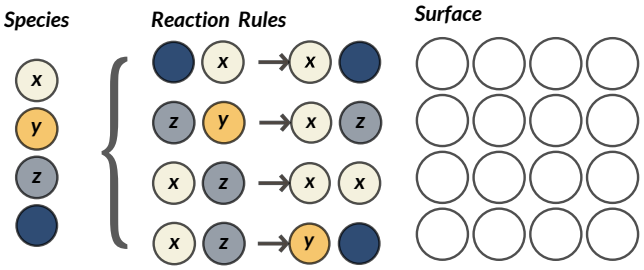


Figure 1 Example sCRN system.

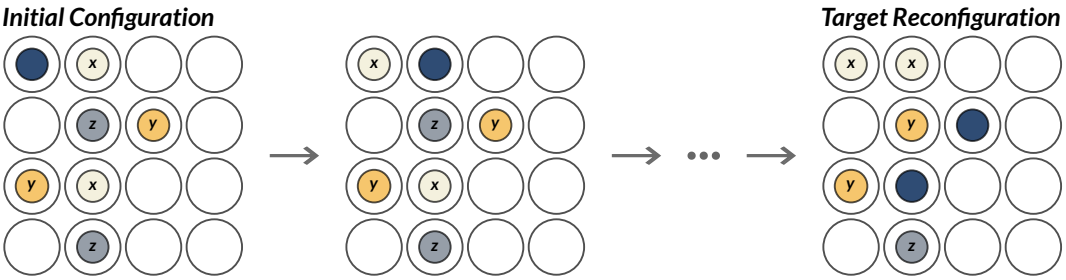


Figure 2 An initial, single step, and target configurations.

Surface, Cell, and Species. A *surface* for a CRN Γ is an (infinite) undirected graph G. The vertices of the surface are called *cells*. A *configuration* is a mapping from each cell to a species from the set S. While our algorithmic results apply to general surfaces, our hardness constructions assume the practical case where G is a grid graph, i.e., an induced subgraph of the infinite square grid (where omitted vertices naturally correspond to cells without any species). When G is an infinite graph, we assume there is some periodic pattern of cells that is repeated on the edges of the surface. Figure 1 shows an example set of species and reactions and a configuration of a surface.

Reaction. A *surface Chemical Reaction Network (sCRN)* consists of a surface and a CRN, where every *reaction* is of the form $A + B \to C + D$ denoting that, when A and B are in neighboring cells, they can be replaced with C and D. A is replaced with C and B with D.

Reachable Configurations. For two configurations I, T, we write $I \to_\Gamma^1 T$ if there exists a $r \in R$ such that performing reaction r on a pair of species in I yields the configuration T. Let $I \to_\Gamma T$ be the transitive closure of $I \to_\Gamma^1 T$, including loops from each configuration to itself. Let $\Pi(\Gamma, I)$ be the set of all configurations T for which $I \to_\Gamma T$ is true. A sequence of reachable states is shown in Figure 2

2.1 Restrictions

Reversible Reactions. A set of reactions R is *reversible* if, for every rule $A + B \to C + D$ in R, the reaction $C + D \to A + B$ is also in R. We may also denote this as a single reversible reaction $A + B \rightleftharpoons C + D$.

Swap Reactions. A reaction of the form $A + B \rightleftharpoons B + A$ is called a *swap reaction*.

k-Burnout. In the k-burnout variant of the model, each vertex of the system's graph can only switch states at most k times (before "burning out" and being stuck in its final state).

2.2 Problems

Reconfiguration Problem. Given a sCRN Γ and two configurations I and T, is $T \in \Pi(\Gamma, S)$?

1-Reconfiguration Problem. Given a sCRN Γ, a configuration I, a vertex v, and a species s, does there exist a $T \in \Pi(\Gamma, S)$ such that T has species s at vertex v?

3 Swap Reactions

In this section, we will show 1-reconfiguration and reconfiguration with swap reactions is PSPACE-complete with only 4 species and 3 swaps in Theorems 3 and 4. We continue by showing that this complexity is tight, that is, reconfiguration with 3 species and swap reactions is tractable in Theorems 5 and 6.

3.1 Reconfiguration is PSPACE-complete

We prove PSPACE-completeness by reducing from the motion planning through gadgets framework introduced in [11]. This is a one player game where the goal is to navigate a robot through a system of gadgets to reach a goal location. The problem of changing the state of the entire system to a desired state has been shown to be PSPACE-complete [1]. This reduction treats the model as a game where the player must perform reactions moving a robot species through the surface.

The Gadgets Framework

Framework. A gadget is a finite set of locations and a finite set of states. Each state is a directed graph on the locations of the gadgets, describing the *traversals* of the gadget. An example can be seen in Figure 3. Each edge (traversal) describes a move the robot can take in the gadget and what state the gadget ends up in if the robot takes that traversal. A robot enters from the start of the edge and leaves at the exit.

In a *system* of gadgets there are multiple gadgets connected by their locations. The *configuration* of a system of gadgets is the state of all gadgets in the system. There is a single robot that starts at a specified location. The robot is allowed to move between connected locations and allowed to move along traversals within gadgets. The system of gadgets can also be restricted to be planar, in which case the cyclic order of the locations on the gadgets is fixed, and the gadgets along with their connections must be embeddable in the plane without crossings.

The *1-player motion planning reachability problem* asks whether there exists a sequence of moves within a system of gadgets which takes the robot from its initial location to a target location. The *1-player motion planning reconfiguration problem* asks whether there exists a sequence of moves which brings the configuration of a system of gadgets to some target configuration.

There are many sets of motion planning models and gadgets to build our reduction. We select 1-player over 0-player since in the sCRN model there are many reactions that may occur and we are asking whether there exists a sequence of reactions which reaches some target configuration; in the same way 1-player motion planning asks if there exists a sequence

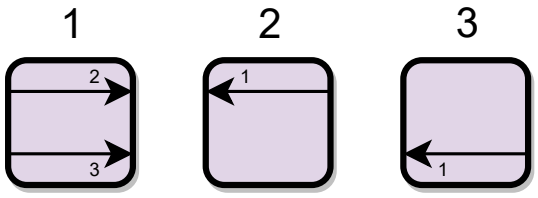


Figure 3 The Locking 2-Toggle (L2T) gadget and its states from the motion planning framework. The numbers above indicate the state and when a traversal happens across the arrows, the gadget changes to the indicated state.

of moves which takes the robot to the target location. The existential query of possible moves/swaps remains the same regardless of whether a player is making decisions vs them occurring by natural processes. The complexity of the gadgets used here are considered in the 0-player setting in [12].

Locking 2-Toggle. The Locking 2-toggle (L2T) is a 4 location, 3 state gadget. The states of the gadget are shown in Figure 3. The L2T has advantages because it universal for reversible deterministic gadgets. Reversibility was important to picking a gadget since swap reactions are naturally reversible.

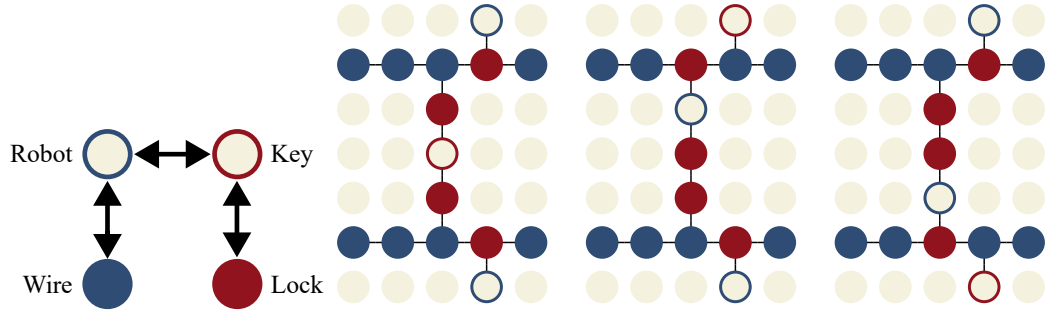
Constructing the L2T

We will show how to simulate the L2T in a swap sCRN system. Planar 1-player motion planning with the L2T was shown to be PSPACE-complete [11]. We now describe this construction.

Species. We utilize 4 species types in this reduction and we name each of them according to their role. First we have the *wire*. The wire is used to create the connection graph between gadgets and can only swap with the robot species. The *robot* species is what moves between gadgets by swapping with the wire and represents the robot in the framework. Each gadget initially contains 2 robot species, and there is one species that starts at the initial location of the robot in the system. The robot can also swap with the key species. Each gadget has exactly 1 *key* species. The key species is what performs the traversal of the gadget by swapping with the lock species. The *lock* species can only swap with the key. There are 4 locks in each gadget. The locks ensure that only legal traversals are possible by the robot species.

These species are arranged into gadgets consisting of two length-5 horizontal tunnels. The two tunnels are connected by a length-3 central vertical tunnel at their 3rd cell. At the 4th cell of both tunnels there is an additional degree 1 cell connected we will call the holding cell.

States and Traversals. The states of the gadget we build are represented by the location of the key species in each gadget. If the key is in the central tunnel of the gadget then we are in state 1 as shown in Figure 4b. Note that in this state the key may swap with the adjacent locks, however we consider these configurations to also be in state 1 and take advantage of this later. The horizontal tunnels of the gadget in this state contain a single lock with an adjacent robot species.



(a) Swap rules/species. **(b)** State 1. **(c)** State 2. **(d)** State 3.

Figure 4 Locking 2-toggle implemented by swap rules. (a) The swap rules and species names. (b-d) The three states of the locking 2-toggle.

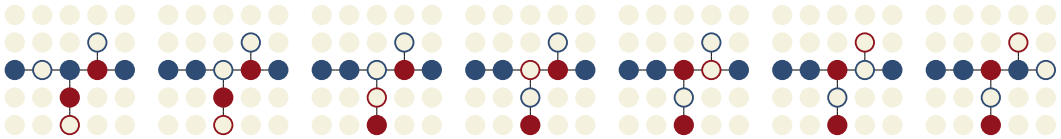


Figure 5 Traversal of the robot species.

States 2 and 3 are reflections of each other (Figures 4c and 4d). This state has a robot in the central tunnel and the key in the respective holding cell. The gadget in this state can only be traversed from right to left in one of the tunnels.

Figure 5 shows the process of a robot species traversing through the gadget. Notice when a robot species "traverse" a gadget, it actually traps itself to free another robot at the exit. We prove two lemmas to help verify the correctness of our construction. The lemmas prove the gadgets we design correctly implement the allowed traversals of a locking 2-toggle.

▶ **Lemma 1.** *A robot may perform a rightward traversal of a gadget through the north/south tunnel if and only if the key is moved from the central tunnel to the north/south holding cell.*

Proof. The horizontal tunnels in state 1 allow for a rightward traversal. The robot swaps with wires until it reaches the third cell where it is adjacent to two locks. However the key in the central tunnel may swap with the locks to reach the robot. The key and robot then swap. The key is then in the horizontal tunnel and can swap to the right with the lock there. It may then swap with the robot in the holding cell. This robot then may continue forward to the right and the key is stuck in the holding cell.

Notice when entering from the left the robot will always reach a cell adjacent to lock species. The robot may not swap with locks so it cannot traverse unless the key is in the central tunnel. ◀

▶ **Lemma 2.** *A robot may perform a leftward traversal of a gadget through the north/south tunnel if and only if the key is moved from the north/south holding cell to the central tunnel.*

Proof. In state 2 the upper tunnel can be traversed and in state 3 the lower tunnel can be traversed. The swap sequence for a leftward traversal is the reverse of the rightward traversal, meaning we are undoing the swaps to return to state 1. The robot enters the gadget and swaps with the key, which swaps with the locks to move adjacent to the central tunnel. The key then returns to the central tunnel by swapping with the robot. The robot species can then leave the gadget to the left.

A robot entering from the right will not be able to swap to the position adjacent to the holding cell if it contains a lock. This is true in both tunnels in state 1 and in the non-traversable tunnels in states 2 and 3. ◀

We use these lemmas to first prove PSPACE-completeness of 1-reconfiguration. We reduce from the planar 1-player motion planning reachability problem.

▶ **Theorem 3.** 1-*reconfiguration is PSPACE-complete with* 4 *species and* 3 *swap reactions or greater even when the surface is a subset of the grid graph.*

Proof. Given a system of gadgets create a surface encoding the connection graph between the locations. Each gadget is built as described above in a state representing the initial state of the system. Ports are connected using multiple cells containing wire species. When more than two ports are connected we use degree-3 cells with wire species. The target cell for 1-reconfiguration is a cell containing a wire located at the target location in the system of gadgets.

If there exists a solution to the robot reachability problem then we can convert the sequence of gadget traversals to a sequence of swaps. The swaps relocate a robot species to the location as in the system of gadgets.

If there exists a swap sequence to place a robot species in the target cell there exists a solution to the robot reachability problem. Any swap sequence either moves an robot along a wire, or traverses it through a gadget. From Lemmas 1 and 2 we know the only way to traverse a gadget is to change its state (the location of its key) and a gadget can only be traversed in the correct state. ◀

Now we show Reconfiguration in sCRNs is hard with the same set of swaps is PSPACE-complete as well. We do so by reducing from the Targeted Reconfiguration problem which asks, given an initial and target configuration of a system of gadgets, does there exist sequence of gadget traversals to change the state of the system from the initial to the target and has the robot reach a target location. Note prior work only shows reconfiguration (without specifying the robot location) is PSPACE-complete[1] however a quick inspection of the proof of Theorem 4.1 shows the robot ends up at the initial location so requiring a target location does not change the computational complexity for the locking 2-toggle. One may also find it useful to note that the technique used in [1] for gadgets and in [17] for Nondeterministic Constraint Logic can be applied to reversible deterministic systems more generally. This means the method described in those could be used to give an alternate reduction directly from 1-reconfiguration of swap sCRNs to reconfiguration of swap sCRNs.

▶ **Theorem 4.** *Reconfiguration is PSPACE-complete with* 4 *species and* 3 *swap reactions or greater.*

Proof. Our initial and target configurations of the surface are built with the robot species at the robots location in the system of gadget, and each key is placed according to the starting configuration of the gadget.

Again as in the previous theorem we know from Lemmas 1 and 2 the robot species traversal corresponds to the traversals of the robot in the system of gadgets. The target surface can be reached if and only the target configuration in the system of gadgets is reachable. ◀

3.2 Polynomial-Time Algorithm

Here we show that the previous two hardness results are tight: when restricting to a smaller cases, both problems become solvable in polynomial time. We prove this by utilizing previously known algorithms for *pebble games*, where labeled pebbles are placed on a subset of nodes of a graph (with at most one pebble per node). A *move* consists of moving a pebble from its current node to an adjacent empty node. These pebble games are again a type of multiplicity friends-and-strangers graph.

▶ **Theorem 5.** *Reconfiguration is in P with* 3 *or fewer species and only swap reactions. Reconfiguration is also in P with* 2 *or fewer swap reactions and any number of species.*

Proof. First we will cover the case of only two swap reactions. There are two possibilities: the two reactions share a common species or they do not. If they do not, we can partition the problem into two disjoint problems, one with only the species involved in the first reaction and the other with only the species from the second reaction. Each of these subproblems has only one reaction, and is solvable if and only if each connected component of the surface has the same number of each species in the initial and target configurations.

The only other case is where we have three species, A, B, and C, where A and C can swap, B and C can swap, but A and B cannot swap. In this case, we can model it as a pebble motion problem on a graph. Consider the graph of the surface where we put a white pebble on each A species vertex, a black pebble on each B species vertex, and leave each C species vertex empty. A legal swap in the surface CRN corresponds to sliding a pebble to an adjacent empty vertex. Goraly et al. [16] gives a linear-time algorithm for determining whether there is a feasible solution to this pebble motion problem. Since the pebble motion problem is exactly equivalent to the surface CRN reconfiguration problem, the solution given by their algorithm directly says whether our surface CRN problem is feasible. ◀

▶ **Theorem 6.** 1*-reconfiguration is in P with* 3 *or fewer species and only swap reactions.* 1*-reconfiguration is also in P with* 2 *or fewer swap reactions.*

Proof. If there are only two swap reactions, we again have two cases depending on whether they share a common species. If they do not share a common species, then we only need to consider the rule involving the target species. The problem is solvable if and only if the connected component of the surface of species involved in this reaction containing the target cell also has at least one copy of the target species. Equivalently, if the target species is A, and A and B can swap, then there must either be A at the target location or a path of B species from the target location to the initial location of an A species.

The remaining case is when we again have three species, A, B, and C, where A and C can swap, B and C can swap, but A and B cannot swap. If C is the target species, then the problem is always solvable as long as there is any C in the initial configuration. Otherwise, suppose without loss of generality that the target species is A. Some initial A must reach the target location. For each initial A, consider the modified problem which has only that single A and replaces all of the other copies of A with B. A sequence of swaps is legal in this modified problem if and only if it was legal in the original problem. The original problem has a solution if and only if any of the modified ones do. We then convert each of these problems to a robot motion planning problem on a graph: place the robot at the vertex with a single copy of A, and place a moveable obstacle at each vertex with a B. A legal move is either sliding the robot to an adjacent empty vertex or sliding an obstacle to an adjacent empty vertex. Papadimitriou et al. [23] give a simple polynomial time algorithm for determining whether it is possible to get the robot to a given target location. By applying their algorithm

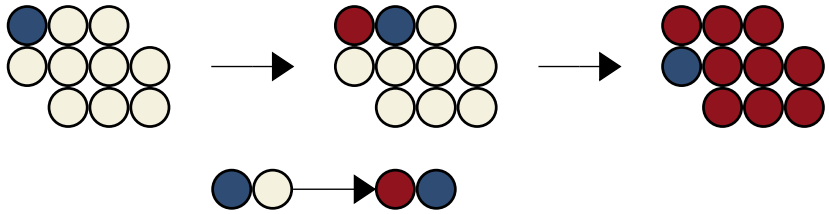


Figure 6 An example reduction from Hamiltonian Path. We are considering graphs on a grid, so any two adjacent locations are connected in the graph. Left: an initial board with the starting location in blue. Middle: One step of the reaction. Right: The target configuration with the ending location in blue. Bottom: the single reaction rule.

to each of these modified problems (one for each cell that has an initial A), we can determine whether any of them have a solution in polynomial time (since there are only linearly many such problems), and thus determine whether the original 1-reconfiguration problem has a solution in polynomial time. ◀

4 Burnout

In this section, we show reconfiguration in 2-burnout with species (A, B, C) and reaction $A + B \to C + A$ is NP-complete in Theorem 7. Next, we show 1-reconfiguration in 1-burnout with 17 species and 40 reactions is NP-complete in Theorem 8.

Reconfiguration and 1-Reconfiguration for burnout sCRNs are in NP since there is the length of any reconfiguration is bounded. For space we do not include this proof but note this has been proved in other system such as Resource Bounded Cellular Automata [13], Freezing Cellular Automata [14] and Freezing Tile Automata [3].

4.1 2-Burnout Reconfiguration

This is a simple reduction from Hamiltonian Path, specifically when we have a stated start and end vertex.

▶ **Theorem 7.** *Reconfiguration in 2-burnout sCRNs with species (A, B, C) and reaction $A + B \to C + A$ is NP-complete even when the surface is a subset of the grid graph. It is also NP-complete with the same species and reactions without the 2-burnout restriction.*

Proof. Let $\Gamma = \{(A, B, C), (A + B \to C + A)\}$. Given an instance of the Hamiltonian path problem on a grid graph H with a specified start and target vertex v_s and v_t, respectively, create a surface G where each cell in G is a node from H. Each cell contains the species B except for the cell representing v_s which contains species A. The target surface has species C in every cell except for the final node containing A, v_t. An example can be seen in Figure 6.

The species A can be thought of as an agent moving through the graph. The species B represents a vertex that hasn't been visited yet, while the species C represents one that has been. Each reaction moves the agent along the graph, marking the previous vertex as visited.

($\Rightarrow$) If there exists a Hamiltonian path, then the target configuration is reachable. The sequence of edges in the path can be used as a reaction sequence moving the agent through the graph, changing each cell to species C finishing at the cell representing v_t.

($\Leftarrow$) If the target configuration is reachable, there exists a Hamiltonian path. The sequence of reactions can be used to construct the path that visits each of the vertices exactly once, ending at v_t.

Note that we have not discussed the effect of Burnout on the reduction. However since each cell transitions through species in the following order: B, A, C this reaction always results in a 2-burnout sCRN so the reduction holds with and without the restriction.

This means the CRN is 2-burnout which bounds the max sequence length for reaching any reachable surface, putting the reconfiguration problem in NP. ◀

4.2 1-Burnout 1-Reconfiguration

For 1-burnout 1-reconfiguration, we show NP-completeness by reducing from 3SAT and utilizing the fact that once a cell has reacted it is burned out and can no longer participate in later reactions.

Figure 7 All the possible configurations of two variable gadgets.

▶ **Theorem 8.** 1-*reconfiguration in* 1-*burnout sCRNs with* 17 *species and* 40 *reactions is NP-complete even when the surface is a subset of the grid graph. It is also NP-complete with the same species and reactions without the* 1-*burnout restriction.*

Proof. We reduce from 3SAT. The idea is to have an "agent" species traverse the surface to assign variables and check that the clauses are satisfied by "walking" through each clause. If the agent can traverse the whole surface and mark the final vertex as "satisfied", there is a variable assignment that satisfies the original 3SAT instance.

Variable Gadget. The variable gadget is constructed to allow for a nondeterministic assignment of the variable via the agent walk. At each intersection, the agent "chooses" a path depending on the reaction that occurs. If the agent chooses "true" for a given variable, it will walk up then walk down to the center species. If the agent chooses "false", the agent will walk down then walk up to the center species. From the center species, the agent can only continue following the path it chose until it reaches the next variable gadget. Examples of the agent assigning variables can be seen in Figure 7.

Each variable assignment is "locked" by way of geometric blocking. When the agent encounters a variable gadget whose variable has already been assigned, the agent must follow that same assignment or it will get "stuck" trying to react with a burnt out vertex. This can be seen in Figure 8.

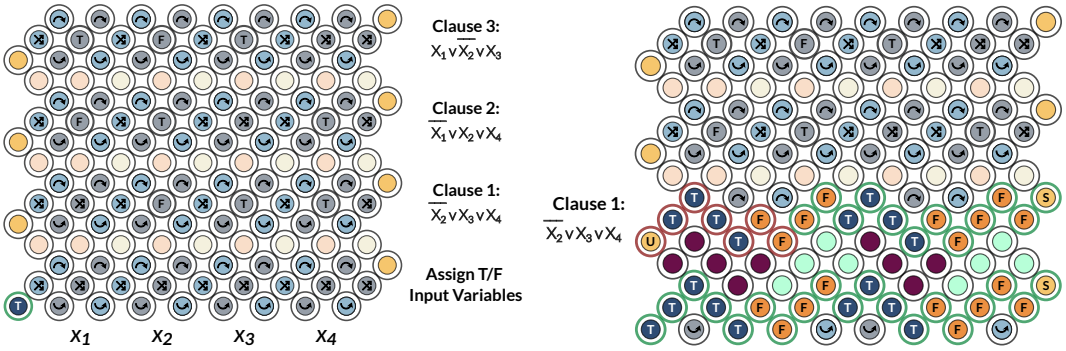
Initial Configuration. First, the configuration is constructed with variable gadgets connected in a row, one for each variable in the 3SAT instance. This row of variable gadgets is where the agent will nondeterministically assign values to the variables. Next, a row of variable gadgets, one row for each clause, is placed on top of the assignment row, connected with helper species to fill in the gaps.

For each clause, if a certain variable is present, the center species of the variable gadget reflects its literal value from the clause. For example, if the variable $x1$ in clause $c1$ should be true to satisfy the clause, the variable gadget representing $x1$ in $c1$'s row will contain a T species in the center cell. Lastly, the agent species is placed in the bottom left of the configuration. An example configuration can be seen in Figure 9.

The agent begins walking and nondeterministically assigns a value to each variable. After assigning every variable, the agent walks right to left. If at an intersection, the agent chooses a different assignment than it did its first pass, the agent becomes "stuck" only being able to react with a burnt out vertex.

(a) Successful navigation of an intersection. **(b)** Agent stuck due to not following the assignment.

Figure 8 The assignment "locking" process.



(a) Example starting configuration. **(b)** The surface after evaluating the first clause.

Figure 9 Reduction from 3SAT to 1-burnout 1-reconfiguration. (a) The starting configuration of the surface for the example formula $\varphi = (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3)$. (b) The configuration after evaluating the first clause. A red outline represents the unsatisfied state, and a green outline represents the satisfied state.

After walking all the way to the left, the first clause can be checked. The agent starts in the unsatisfied state, walking through each variable in the row, left to right. If the current variable assignment at a variable gadget satisfies this clause, the agent changes to the satisfied state and continues walking. If the agent walks through all the variables without becoming satisfied, the computation ends. If the clause was satisfied, the agent continues by walking back, right to left, to begin evaluation of the next clause. If the agent walks all the way to the final vertex with a satisfied state, then the initial variable assignment satisfies all the clauses.

($\Rightarrow$) If there exists a variable assignment that satisfies the 3SAT instance, then the final vertex can be marked with the satisfied state s. The agent can only mark the final cell with the satisfied state s if all clauses can be satisfied.

($\Leftarrow$) If the final vertex can be marked with satisfied state s, there exists a variable assignment that satisfies the 3SAT instance. The variable assignment that the agent non-deterministically chose can be read and used to satisfy the 3SAT instance. ◀

5 Single Reaction

When limited to a single reaction, we show a complete characterization of the reconfiguration problem. There exists a reaction using 3 species for which the problem is NP-complete. For all other cases of 1 reaction, the problem is solvable in polynomial time.

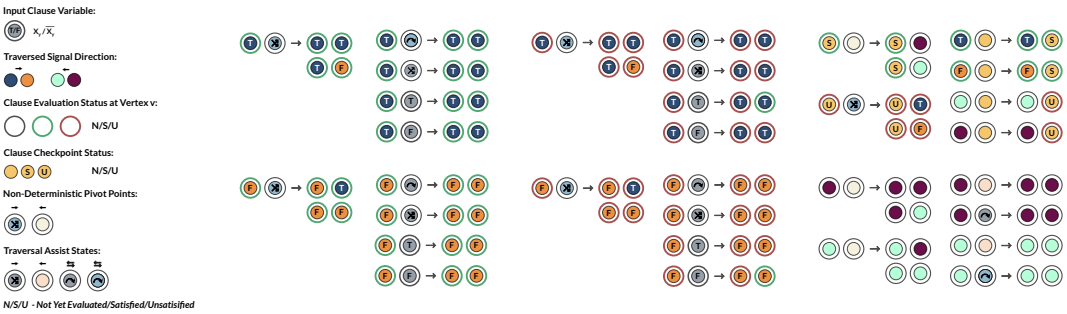


Figure 10 Species identification and transition rules for 1-burnout 1-reconfiguration.

5.1 2 Species

We start with proving reconfiguration is in P when we only have 2 species and a single reaction.

▶ **Lemma 9.** *Reconfiguration with species* $\{A, B\}$ *and reaction* $A + A \to A + B$ *OR* $A + B \to A + A$ *is solvable in polynomial time on any surface.*

Proof. The reaction $A + B \to A + A$ is the reverse of the first case. By flipping the target and initial configurations, we can reduce from reconfiguration with $A + B \to A + A$ to reconfiguration $A + A \to A + B$.

We now solve the case where we have the reaction $A + A \to A + B$.

All cells that start and end with species B can be ignored as they do not need to be changed, and can not participate in any reactions. If there is a cell that contains B in the initial configuration but A in the target, the instance is "no" as B may never become A.

Let any cell that starts in species A but ends in species B be called a *flip* cell, and any species that starts in A and stays in A a *catalyst* cell.

An instance of reconfiguration with these reactions is solvable if and only if there exists a set of spanning trees, each rooted at a catalyst cell, that contain all the flip cells. Using these trees, we can construct a reaction sequence from post-order traversals of each spanning tree, where we have each non-root node react with its parent to change itself to a B. In the other direction, given a reaction sequence, we can construct the spanning trees by pointing each flip cell to the neighbor it reacts with. ◀

▶ **Lemma 10.** *Reconfiguration with species* $\{A, B\}$ *and reaction* $A + A \to B + B$ *is solvable in polynomial time on any surface.*

Proof. Reconfiguration in this case can be reduced to perfect matching. Create a graph M including a node for each cell in S containing the A species initially and containing B in the target, with edges between nodes of neighboring cells. If M has a perfect matching, then each edge in the matching corresponds to a reaction that changes A to B. If the target configuration is reachable, then the reactions form a perfect matching since they include each cell exactly once. ◀

▶ **Theorem 11.** *Reconfiguration with* 2 *species and* 1 *reaction is in P on any surface.*

Proof. As we only have two species and a single reaction, we can analyze each of the four cases to show membership in P. We divide into two cases:

***A* + *A*:** When a species reacts with itself, it can either change both species, which is shown to be in P by Lemma 10; or it changes only one of the species, which is in P by Lemma 9.

***A* + *B*:** When two different species react, they can either change to the same species, which is in P by Lemma 9; or they can both change, which is a swap and thus is in P by Theorem 5. ◀

5.2 3 or more Species

Moving up to 3 species and 1 reaction, we showed earlier that there exists a reaction for which reconfiguration is NP-complete in Theorem 7. Here, we give reactions for which reconfiguration between 3 species is in P, and in Corollary 15 we prove that all remaining reactions are isomorphic to one of the reactions we've analyzed.

▶ **Lemma 12.** *Reconfiguration with species* (A, B, C) *and reaction* $A + B \to C + C$ *is solvable in polynomial time on any surface.*

Proof. At a high level, we create a new graph of all the cells that must change to species C, and add an edge when the two cells can react with each other. Since a reaction changes both cells to C we can think of the reaction as "covering" the two reacting cells. Finding a perfect matching in this new graph will give a set of edges along which to perform the reactions to reach the target configuration.

Consider a surface G and a subgraph $G' \subseteq G$ where we include a vertex v' in G' for each cell that contain A or B in the initial configuration and C in the target configuration. We include an edge (u', v') between any vertices in G' that contain different initial species, i.e. any pair of cell which one initially contains A and the other initially B.

Reconfiguration is possible if and only if there is a perfect matching in G'. If there is a perfect matching then there exists a set of edges which cover each cell once. Since G' represents the cells that must change states, and the edges between them are reactions, the covering can be used as a sequence of pairs of cells to react. If there is a sequence of reactions then there exists a perfect matching in G': each cell only reacts once so the matching must be perfect, and the cells that react have edges between them in G'. ◀

▶ **Lemma 13.** *Reconfiguration with species* (A, B, C) *and reaction* $A + B \to A + C$ *is solvable in polynomial time on any surface.*

Proof. The instance of reconfiguration is solvable if and only if any cell that ends with species C either contained C in the initial configuration, or started with species B and have an A adjacent to perform the reaction. Additionally, since a reaction cannot cause a cell to change to A or B, each cell with an A or B in the target configuration must contain the same species in the initial configuration. ◀

The final case we study is 4 species 1 reaction. Any sCRN with 5 or more species and 1 reaction has a species which is not included in the reaction.

▶ **Lemma 14.** *Reconfiguration with species* (A, B, C, D) *and the reaction* $A + B \to C + D$ *is in P on any surface.*

Proof. We can reduce Reconfiguration with $A + B \to C + D$ to perfect matching similar to Lemma 12. Create a new graph with each vertex representing a cell in the surface that must change species. Add an edge between each pair of neighboring cells that can react (between one containing A and the other B). A perfect matching then corresponds to a sequence of reactions that changes each of the species in each cell to C or D. ◀

▶ **Corollary 15.** *Reconfiguration with* 3 *or greater species and* 1 *reaction is NP-complete on any surface.*

Proof. First, from Theorem 7 we see that there exists a case of reconfiguration with 3 species that is NP-hard with or without the burnout restriction.

For membership in NP, we analyze each possible reaction. We note that we only need to consider two cases for the left hand side of the rule, $A + A$ and $A + B$. Any other reaction is isomorphic to one of this form as we can relabel the species. For example, rule $B+C \to A+A$ can be relabeled as $A + B \to C + C$. Also, we know that C must appear somewhere in the right hand side of the rule. If it does not then the reaction only takes place between two species, which is always polynomial time as shown above, or it involves a species we can relabel as C.

Here are the cases for $A + B$ and our analysis results:

$A + B \to A + C$	P in Lemma 13
$A + B \to C + B$	P in Lemma 13 under isomorphism
$A + B \to C + A$	NP in Theorem 7
$A + B \to B + C$	NP in Theorem 7 under isomorphism
$A + B \to C + C$	P in Lemma 12
$A + B \to C + D$	P in Lemma 14

When we have $A + A$ on the left side of the rule, the only case we must consider is $A + A \to B + C$ (since all 3 species must be included in the rule). We have already solved this reaction: first swap the labels of A and C giving rule $C + C \to B + A$, then reverse the rule to $B + A \to C + C$ and swap the initial and target configuration. Finally since rules do not care about orientation this is equivalent to the rule $A + B \to C + C$ in Lemma 12.

Finally, for 4 species and greater, the only new case is $A + B \to C + D$, which is proven to be in P in Lemma 14. Any other case would have species that are not used since a rule can only have 4 different species in it.

Thus, all cases are either in NP, or in P which is a subset of NP, therefore, the problem is in NP. Also, since our results for each case apply for any surface, the same is true in general. ◀

6 Conclusion

In this paper, we explored the complexity of the configuration problem within natural variations of the surface CRN model. While general reconfiguration is known to be PSPACE-complete, we showed that it is still PSPACE-complete even with several extreme constraints. We first considered the case where only swap reactions are allowed, and showed reconfiguration is PSPACE-complete with only four species and three distinct reaction types. We further showed that this is the smallest possible number of species for which the problem is hard by providing a polynomial-time solution for three or fewer species when only using swap reactions.

We next considered surface CRNs with rules other than just swap reactions. First, we considered the burnout version of the reconfiguration problem, and then followed by the normal version with small species counts. In the case of 2-burnout, we showed reconfiguration is NP-complete for three species and one reaction type, and 1-burnout is NP-complete for 17 species with 40 distinct reaction types. Without burnout, we achieved, as a corollary, that three species, one reaction type is NP-complete while showing that dropping the species count down to two yields a polynomial-time solution.

6.1 Computing Polynomial Space Functions

An interpretation of Theorem 3 is that surface Chemical Reactions are capable of computing any function that can be computed in polynomial space. Perhaps the most important PSPACE-Complete is the acceptance problem for polynomial space Turing machines. While there may be a few reduction between these problems, we can may turn any polynomial space Turing machine into a surface CRN such that the robot species swaps with a wire species at a target location. In experiments one can imagine the target location as having a special type of wire species that acts as a reporting, emitting a signal when it reacts with the robot species. The size of the surface is polynomial in the space of the Turing machine since these are all polynomial time reductions. While we do not claim this experiment can be done with such a small number of species, but rather that theoretically more sequence efficient reaction systems which can compute should exists by taking advantage of the surface.

Our polynomial time algorithms describe experiments with 1, 2, or 3 reactions on surfaces where well studied algorithms for problems such as matching and motion planning may be of use.

6.2 Open Problems

This work introduced new concepts that leaves open a number of directions for future work. While we have fully characterized the complexity of reconfiguration for the swap-only version of the model, the complexity of reconfiguration with general rule types for three species systems remains open if the system uses more than one rule. All of hardness results also use a square grid graph, while our algorithms work on general surfaces. We would like to know if the threshold for hardness can be lowered on more general graphs. In the 1-burnout variant of the model, we have shown 1-reconfiguration to be NP-complete, but the question of general reconfiguration remains a "burning" open question.

References

1. Joshua Ani, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Traversability, reconfiguration, and reachability in the gadget framework. In *WALCOM: Algorithms and Computation: 16th International Conference and Workshops, WALCOM 2022, Jember, Indonesia, March 24–26, 2022, Proceedings*, pages 47–58. Springer, 2022.
2. Tatiana Brailovskaya, Gokul Gowri, Sean Yu, and Erik Winfree. Reversible computation using swap reactions on a surface. In *International Conference on DNA Computing and Molecular Programming*, pages 174–196. Springer, 2019.
3. David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Verification and computation in restricted tile automata. *Natural Computing*, pages 1–19, 2020.
4. Cameron Chalk, Austin Luchsinger, Eric Martinez, Robert Schweller, Andrew Winslow, and Tim Wylie. Freezing simulates non-freezing tile automata. In *DNA Computing and Molecular Programming: 24th International Conference, DNA 24, Jinan, China, October 8–12, 2018, Proceedings 24*, pages 155–172. Springer, 2018.
5. Gourab Chatterjee, Neil Dalchau, Richard A. Muscat, Andrew Phillips, and Georg Seelig. A spatially localized architecture for fast and modular DNA computing. *Nature nanotechnology*, 12(9):920–927, 2017.
6. Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural computing*, 13:517–534, 2014.
7. Samuel Clamons, Lulu Qian, and Erik Winfree. Programming and simulating chemical reaction networks on a surface. *Journal of the Royal Society Interface*, 17(166):20190790, 2020.

8 Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. In *Algorithmic bioprocesses*, pages 543–584. Springer, 2009.

9 Frits Dannenberg, Marta Kwiatkowska, Chris Thachuk, and Andrew J Turberfield. DNA walker circuits: computational potential, design, and verification. *Natural Computing*, 14(2):195–211, 2015.

10 Colin Defant and Noah Kravitz. Friends and strangers walking on graphs. *Combinatorial Theory*, 1, 2021.

11 Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *9th International Conference on Fun with Algorithms (FUN 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

12 Erik D Demaine, Robert A Hearn, Dylan Hendrickson, and Jayson Lynch. Pspace-completeness of reversible deterministic systems. In *Machines, Computations, and Universality: 9th International Conference, MCU 2022, Debrecen, Hungary, August 31–September 2, 2022, Proceedings*, pages 91–108. Springer, 2022.

13 Alberto Dennunzio, Enrico Formenti, Luca Manzoni, Giancarlo Mauri, and Antonio E Porreca. Computational complexity of finite asynchronous cellular automata. *Theoretical Computer Science*, 664:131–143, 2017.

14 Eric Goles, Diego Maldonado, Pedro Montealegre, and Martín Ríos-Wilson. On the complexity of asynchronous freezing cellular automata. *Information and Computation*, 281:104764, 2021.

15 Eric Goles, Nicolas Ollinger, and Guillaume Theyssier. Introducing freezing cellular automata. In *Cellular Automata and Discrete Complex Systems, 21st International Workshop (AUTOMATA 2015)*, volume 24, pages 65–73, 2015.

16 Gilad Goraly and Refael Hassin. Multi-color pebble motion on graphs. *Algorithmica*, 58:610–636, 2010.

17 Robert A Hearn and Erik D Demaine. *Games, puzzles, and computation*. CRC Press, 2009.

18 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

19 Aleksa Milojevic. Connectivity of old and new models of friends-and-strangers graphs. *arXiv preprint arXiv:2210.03864*, 2022.

20 Kenichi Morita and Yoshikazu Yamaguchi. A universal reversible turing machine. In *Machines, Computations, and Universality: 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007. Proceedings 5*, pages 90–98. Springer, 2007.

21 Richard A. Muscat, Karin Strauss, Luis Ceze, and Georg Seelig. DNA-based molecular architecture with spatially localized components. *ACM SIGARCH Computer Architecture News*, 41(3):177–188, 2013.

22 Jennifer Padilla, Wenyan Liu, and Nadrian Seeman. Hierarchical self assembly of patterns from the robinson tilings: Dna tile design in an enhanced tile assembly model. *Natural computing*, 11:323–338, June 2012. doi:10.1007/s11047-011-9268-7.

23 Christos H Papadimitriou, Prabhakar Raghavan, Madhu Sudan, and Hisao Tamaki. Motion planning on a graph. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 511–520. IEEE, 1994.

24 Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universität Bonn, 1962.

25 Lulu Qian and Erik Winfree. Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface. In *DNA Computing and Molecular Programming: 20th International Conference, DNA 20, Kyoto, Japan, September 22-26, 2014. Proceedings*, volume 8727, page 114. Springer, 2014.

26 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *natural computing*, 7:615–633, 2008.

27 David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

28 Guillaume Theyssier and Nicolas Ollinger. Freezing, bounded-change and convergent cellular automata. *Discrete Mathematics & Theoretical Computer Science*, 24, 2022.

29 Anupama J. Thubagere, Wei Li, Robert F. Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, et al. A cargo-sorting DNA robot. *Science*, 357(6356):eaan6558, 2017.

30 Damien Woods and Turlough Neary. The complexity of small universal turing machines: A survey. *Theoretical Computer Science*, 410(4-5):443–450, 2009.