Modelling and verification for DNA nanotechnology

Frits Gerrit Willem Dannenberg

Balliol College

University of Oxford

A thesis submitted for the Degree of

Doctor of Philosophy

Hilary 2016

Abstract

DNA nanotechnology is a rapidly developing field that creates nanoscale devices from DNA, which enables novel interfaces with biological material. Their therapeutic use is envisioned and applications in other areas of basic science have already been found. These devices function at physiological conditions and, owing to their molecular scale, are subject to thermal fluctuations during both preparation and operation of the device. Troubleshooting a failed device is often difficult and we develop models to characterise two separate devices: DNA walkers and DNA origami. Our framework is that of continuous-time Markov chains, abstracting away much of the underlying physics. The resulting models are coarse but enable analysis of system-level performance, such as 'the molecular computation eventually returns the correct answer with high probability'. We examine the applicability of probabilistic model checking to provide guarantees on the behaviour of nanoscale devices, and to this end we develop novel model checking methodology.

We model a DNA walker that autonomously navigates a series of junctions, and we derive design principles that increase the probability of correct computational output. We also develop a novel parameter synthesis method for continuous-time Markov chains, for which the synthesised models guarantee a predetermined level of performance. Finally, we develop a novel discrete stochastic assembly model of DNA origami from first principles. DNA origami is a widespread method for creating nanoscale structures from DNA. Our model qualitatively reproduces experimentally observed behaviour and using the model we are able to rationally steer the folding pathway of a novel polymorphic DNA origami tile, controlling the eventual shape.

Acknowledgements

Prof. Marta Kwiatkowska and Prof. Andrew J. Turberfield have supervised me over the course of my DPhil, and I thank each for their persistent interest in my research. This thesis would not have been possible without them, for which I am in perpetual debt.

I thank my scientific collaborators, Jon Bath, Milan Češka, Katherine Dunn, Ernst Moritz Hahn, Tom Ouldridge, Nicola Paoletti, and Chris Thachuk, each of whom it was a pleasure to work with. My thanks go to Dave Parker and Vojtěch Forejt for their advice, and their hospitality in setting me up with a subversion repository. Andrew Phillips of Microsoft Research Cambridge has kindly hosted me twice in his lab, and I thank him for the collaboration and the memorable summers. Finally, here I acknowledge the other students in the Kwiatkowska group, whose presence made working on this project that much more enjoyable.

Research grants by

Microsoft Research PhD Scholarship,
ERC grant VERIWARE,
Prins Bernhard Cultuurfonds (Niemans-Schootemeijer).

Contents

List of Figures

Page

List of Tables

Page

Chapter 1

Introduction

DNA is found in the cell nucleus, where it makes up the genetic material of the cell. Now more accessible than ever before, DNA strands are sold commercially in a made-to-order fashion. By designing strands of DNA to interact with each other, complex devices are created by combining them in a reaction tube. Formidable achievements include tiny artificial motors, nanoscale structures of virtually any shape and DNA computers that evaluate Boolean formulas. These artificial devices interact naturally with biological materials and one proposed application is that of medical diagnostics, where single-use devices interact directly with biological samples to detect disease. A unifying characteristic of these devices is that of self-assembly: these devices are created by simply mixing together DNA strands. This is in contrast to, for instance, photolithography, a top-down process which imprints nanoscale details onto surfaces via macroscopic photo masks.

Despite its frequent use, we have only just begun to understand DNA as an engineering material. As the applications of DNA nanotechnology develop further, so does the need for detailed understanding of their operation. Crucially, to make the transition from prototype to application, a device must function reliably, in spite of the thermal fluctuations that occur both during preparation and operation of the device. Troubleshooting a failed device is difficult because of the limited options to directly observe its operation. Typically, partial execution is characterised in control experiments, and new insights are then incorporated in the preparation of the actual device. In addition, the non-covalent binding of DNA is intricate and DNA hybridization reactions are relatively difficult to characterise, especially when taking into account variations in temperature and mechanical stress.

In this thesis we model two existing DNA devices, predict their operation and identify modes of error, and enable rational design of their execution. We obtain significant new insights into the self-assembly of DNA origami, a wide-spread technique to create structures from DNA, which we validate through experimental observation. Specifically, we create the first-ever dis-

crete stochastic self-assembly model of DNA origami and demonstrate the existence of folding pathways. By clever manipulation of the design we steer a polymorphic tile to fold into various shapes. Reminiscent of the Anfinsen experiment, a breakthrough study on protein folding, we find that stable long range interactions are highly significant to the folding pathway, and our modifications primarily concern adjustments to these hyper-stable design elements.

We also examine the applicability of probabilistic model checking to obtain predictions from our models. Probabilistic model checking is the study of automated verification methods to compute the probability of a stochastic system to meet a given specification. Specifications are given in a temporal logic and algorithms enable the automated analysis of the validity of these properties. Formal verification methods were originally conceived with the purpose of determining the fault-free operation of software. Given the stochastic nature of molecular devices, the application of probabilistic verification to DNA nanotechnology seems promising. We aim for analysis of system-level performance, such as 'the molecular computation eventually returns the correct answer with high probability'.

In Chapter 4 we find that exhaustive probabilistic model checking is viable for a limited range of models, and instead we frequently employ statistical model checking (based on stochastic simulation) to obtain model predictions. In Chapter 6 and 7, where the self-assembly of DNA origami is modelled, we do away with properties specified in temporal logic and permit domain-specific properties that consider event precedence. Despite that, we attempt to increase the applicability of probabilistic model checking methods to DNA nanotechnology by developing new methodology in Chapter 4. In specific we develop a parameter synthesis method that guarantees the correct behaviour for any parameter in the synthesised set, which is a significant improvement over previous work.

1.1 Contribution

We summarise the novel results of this thesis below, and list the author's involvement in Section 1.2.

- DNA walkers are man-made molecular motors that travel along a nanoscale track of DNA, mimicking the function of transport proteins found in nature, such as dynein and kinesin. We show that, for a specific and previously published type of walker, tracks can be used to encode Boolean formulas. We demonstrate the efficient embedding of any 3-conjunctive normal formula as a walker circuit. We develop a continuous-time Markov chain (CTMC) model of the stepping behaviour in PRISM, calibrate it to pre-existing measurements, and demonstrate the use of probabilistic model checking methods to detect modes of error. The

model files and scripts created for this study are made public and can be found, including a detailed description, at [1]. Two of the model scripts are included as Appendix A.

- Standard uniformisation is an often-used numerical technique underlying the model checking of time-bounded properties for CTMC models. We extend the fast adaptive uniformisation method, a generalisation of standard uniformisation intended to increase scalability, to compute time-bounded cumulative reward properties.

- We develop a method for precise parameter synthesis so that, given a CTMC model specified over unknown parameters and a property specified in continuous stochastic logic (CSL), parameters are identified for which the model is guaranteed to satisfy the property. This is especially useful for biologically inspired reaction networks, where reaction rate constants may depend on temperature or other environmental variables.

- DNA origami is a frequently used technique to create self-assembling nanoscale structures from plasmid-derived circular DNA and short staple strands. We develop a thermodynamically consistent CTMC model of origami self-assembly by finding an efficient and self-consistent way to evaluate the contribution to the free energy of the system that the geometric constrains induce by staple links. Looping constraints within partially folded structures are interpreted as simple cycles in an embedded graph. We derive a functional expression for loop stability that we calibrate to existing models of isolated bulge loop formation. The model predicts folding properties such as melting temperature and hysteresis well within the expected ranges.

- We demonstrate detailed understanding and control over the folding pathway of a DNA origami. The folding of a polymorphic origami tile was simulated using the self-assembly model, and was found to strongly depend on staple design. The folding pathway was efficiently manipulated through rational design, using minor changes in the staple domains. The code created to simulate the self-assembly, including settings to simulate the polymorphic tile, is public and can be accessed at [2].

1.2 Publications

Select material from this thesis is derived from joint-authored papers, and here I describe my involvement beyond writing.

The computational expressiveness and modelling of a DNA walker was considered in [3, 4]. In this work I collaborated with my co-authors to determine abstractions and the reporting strategy. I developed and analysed the CTMC model of the DNA walker in consultation with

co-authors. I created a custom tool to generate model and property files for two probabilistic model checking tools (PRISM and MARCIE) based on textual description of the circuits. The construction to embed 3CNF formula as walker circuits is due to Dr. Chris Thachuk. Chapter 4 contains the results from this study.

In [5, 6] the derivation of cumulative rewards for the fast adaptive uniformisation method is presented. I created a prototype implementation of the fast adaptive uniformisation method, which was applied to simulate the self-assembly model of Chapter 6, and assisted in the subsequent re-implementation of the method in the PRISM probabilistic model checking tool. Section 5.1 contains results from this study. The proof strategy in Thm. 5.1.4 is due to Dr. Ernst Moritz Hahn.

In [7, 8] a precise parameter synthesis method for CTMC models was developed. I collaborated to develop the problem definition, model-checking algorithms, stability analysis and heuristics. Section 5.2 contains results from this study.

In [9] a thermodynamically consistent CTMC model of the self-assembly of DNA origami is developed. The 'local' model and its computational methodology presented here was developed and implemented by me under the supervision of Prof. Andrew J. Turberfield. The subsequent development of the 'global' model, and the calibration to the SantaLucia model of bulge loops, occurred in collaboration with Dr. Thomas E. Ouldridge, who also provided the thermodynamic interpretation of both the local and global model. The computational methodology of the global model was worked out in consultation with Thomas and all simulation code was written by me. I collaborated in the analysis of the model results. The simulation software is publicly available at [2]. Chapter 6 contains results from this study.

In [10] the folding pathways of a polymorphic DNA origami tile are described. Based on the 'local' self-assembly model of Chapter 6, minor adjustments to the tile design are found to result in significant shifts in the distribution of shapes: this was confirmed in experiment. I contributed to the analysis of the possible strain-free shapes of the polymorphic tile. I designed and implemented the exclusion algorithm and analysed folding pathways *in silico* under the supervision of Prof. Andrew J. Turberfield. I collaborated in describing these folding pathways. The half-seam and elongated staple modifications were suggested by me. All simulation code was written by me and all model predictions were obtained from my code. The simulation software is publicly available at [2]. The design, synthesis and AFM imaging of the origami tile, and subsequent fitting of shapes, is due to Dr. Katherine E. Dunn. Chapter 6 contains results from this study.

1.3 Thesis structure

This thesis is structured as follows. Chapter 2 is an introduction to DNA nanotechnology. Chapter 3 is a background chapter on probabilistic model checking and provides essential definitions for the remainder of this thesis. Chapter 4 describes a model of a DNA walker, for which we explore its application as a computational device. We identify modes of error in its execution and develop design principles to maximise performance. Chapter 5 describes two novel contributions related to model checking for continuous-time Markov chains. Section 5.1 develops fast adaptive uniformisation for reward properties, which we apply to the DNA walker model. In Section 5.2 we describe a novel parameter synthesis method which permits the model to contain imprecisely specified model parameters. Chapter 6 describes a novel method of modelling and analysing the self-assembly of DNA origami, a frequently used technique to create nanoscale structures from DNA. Chapter 7 applies the 'local' self-assembly model from Chapter 6 to a novel polymorphic DNA origami tile. Through modifications to the design we demonstrate control over its folding pathway. Chapter 8 concludes the thesis and we discuss the direction of future research.

Chapter 2

Introduction to DNA nanotechnology

A DNA nanodevice is created by combining DNA strands in solution, where each type of strand has a specific sequence over the alphabet of bases: A, T, C and G. The interaction between strands is controlled by manipulating the strands sequences, as hybridization is only possible between strands with complementary sequences. Some devices are intended to operate dynamically, for example a cascade of reactions is triggered after input is added [11, 12]. For other devices their form *is* the function, for example, synthetic membrane channels were constructed from DNA (Fig. 2.1 and [13, 14]).

We now review the main concepts relevant to DNA nanotechnology that feature in the thesis. Starting in Section 2.6 we discuss literature on DNA nanotechnology.

2.1 Deoxyribonucleic acid

Deoxyribonucleic acid (DNA) is a linear polymer composed of nucleotides. Each nucleotide (nt) consists of a base and a phosphate-deoxyribose backbone group, where the base is one of four variants: Adenine (A), Thymine (T), Guanine (G) and Cytosine (C). A duplex of two strands is formed when the bases align in a complementary fashion, forming A-T and C-G pairs, as in Fig. 2.2. Each strand of DNA has an orientation, labelled from the $5'$ to the $3'$ end, and pairing is only possible when the strands run in opposite direction. The paired form is called double-stranded DNA (dsDNA), and involves the two strands coiling around one another in a helical pattern. The stacking of bases forms a hydrophobic core surrounded by the negatively charged sugar-phosphate backbone and the structure is stabilized by the presence of cations that screen electrostatic self-interaction. The precise shape of the helical stacking depends on environmental conditions, permitting compressed geometries (A-DNA) or stretched (Z-DNA)

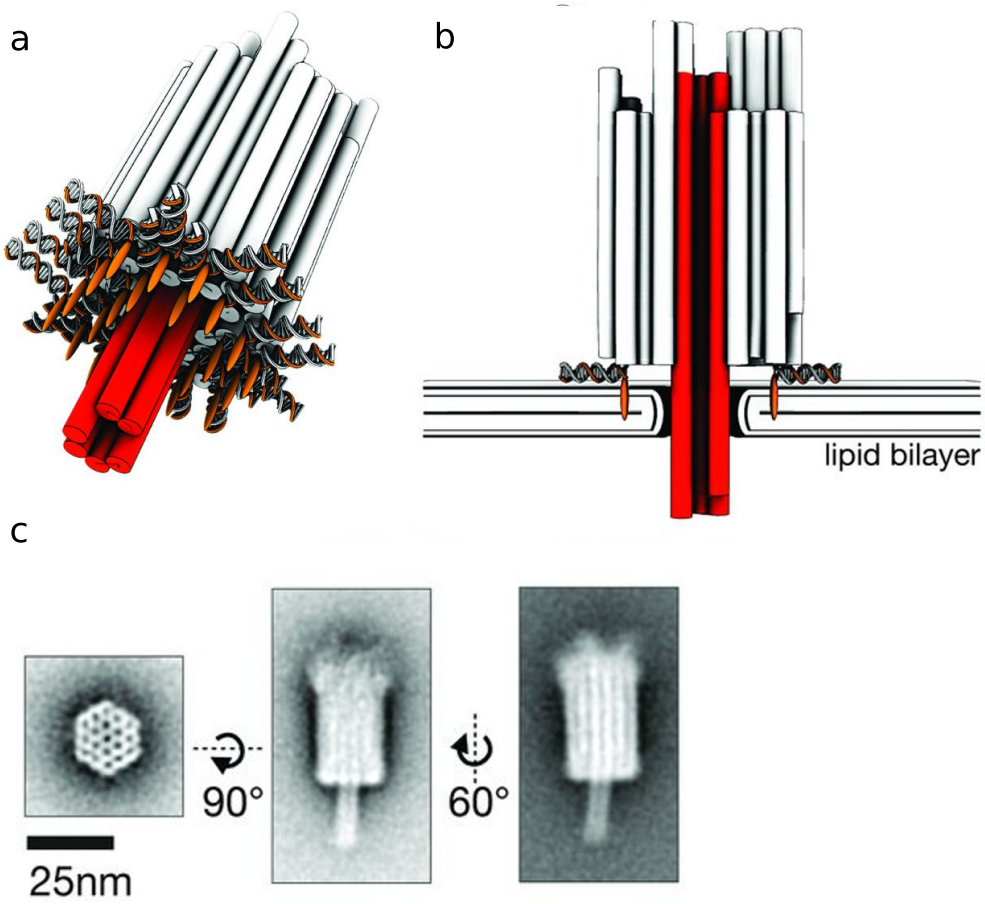


Figure 2.1: Reproduced from [14] with permission. A synthetic membrane channel made from DNA origami. a) Diagram of the DNA structure, which contains cholesterol-modified oligonucleotides (orange) around the base. b) The structure is designed to attach and pierce lipid bilayers. The shaft (in red) then forms an unblocked channel through the membrane. c) After preparation, purification and treatment with a staining dye, the nanopores are imaged using Transfer Electron Microscopy (TEM). Class averages of the recorded images are shown here.

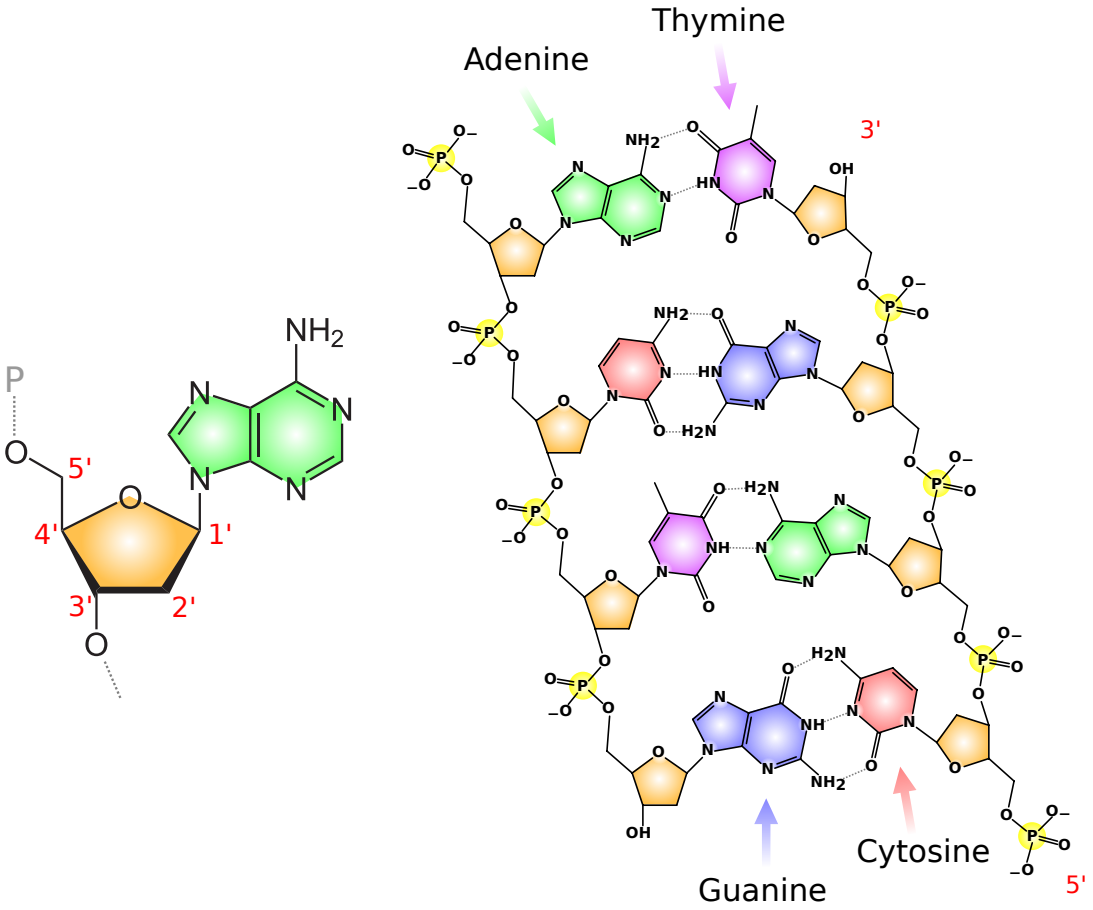


Figure 2.2: DNA is a polymer of nucleotides. Left: The directionality of DNA is established by enumerating the carbon chain in the deoxyribose (sugar) group. Right: Two complementary strands of DNA hybridize through non-covalent interaction. Modified from M.P. Ball (Creative Commons).

forms. Under physiological conditions DNA occurs in the right-handed B-conformation, as in Fig. 2.3, for which the rise per base-pair is approximately 0.34 nm and the width of the helix is approximately 2 nm. The helical wrapping results in a major and minor groove between the phosphate linkers, and the periodicity of the helix is approximately 10.5 base-pairs per turn. Self-complementary DNA can form secondary structures such as a so-called 'hairpin loop' or 'stem-loop' (Fig. 2.4a).

DNA is traditionally studied for its essential role as a carrier of information within the cell and its role in reproduction. Expression of traits in organisms were first described as a function of the organisms' ancestry by Gregor Mendel circa 1865. DNA itself was first isolated in 1869 by Friedrich Miescher, and in 1928 it was discovered through Frederick Griffith's experiment that bacteria can transform through the absorption of genetic material. Later it was found that DNA, as opposed to protein, was responsible for the transformation, through the Avery-MacLeod-McCarty experiment. The double helical structure of DNA was discovered by Watson and Crick in 1953.

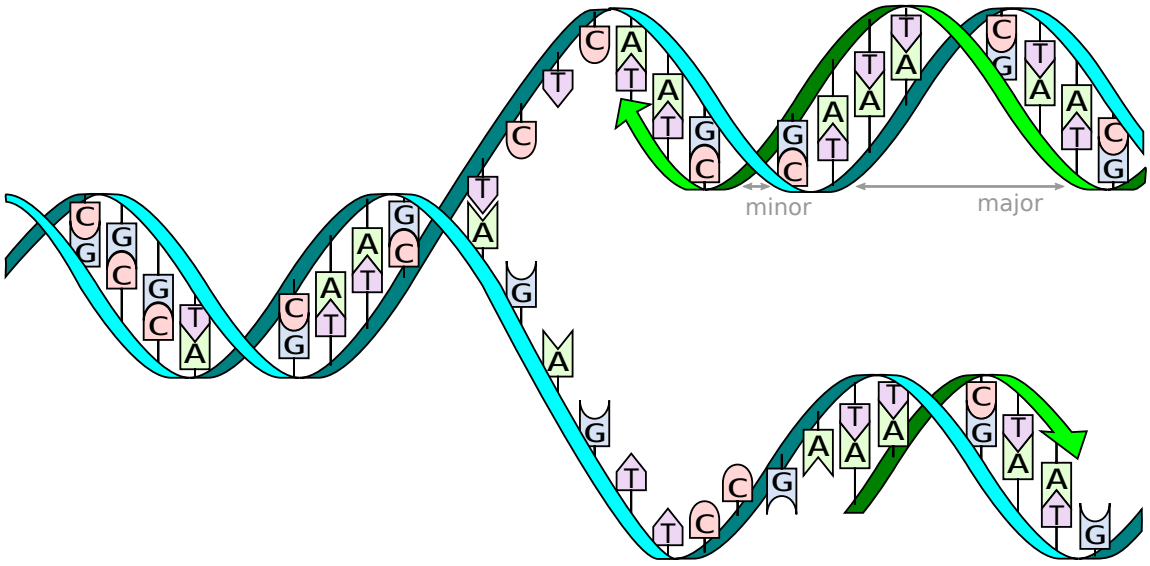


Figure 2.3: Schematic representation of double-stranded DNA in B-conformation. The helix on the left (blue) splits halfway and the two strands are hybridized by separate green strands. Also shown are the major and minor grooves. Modified from M.P. Ball (Creative Commons).

2.2 DNA synthesis

DNA strands consisting of up to 200 nucleotides, called oligonucleotides or oligos for short, are available commercially, where they are produced by chemical synthesis. During production each nucleotide is incorporated with some probability of error and as a result longer sequences are harder to produce. Consequently, orders for longer sequences are charged at a premium.

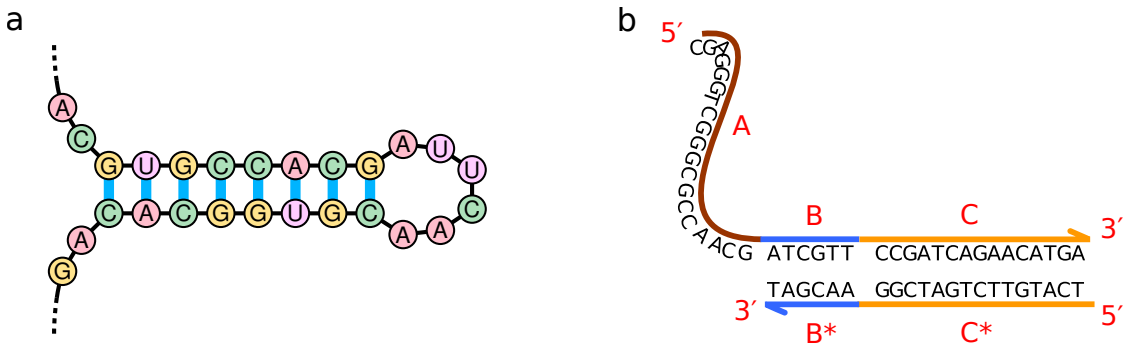


Figure 2.4: a) Self-complementary DNA in hairpin formation. (Creative Commons, anonymous) b) A domain-level description is an efficient abstraction of DNA. In this figure the top strand consists of the domains, A, B and C. The bottom strand consists of domains C* and B*.

Sequences with a strong secondary structure, such as G-quadruplexes, are also harder to synthesise[1]. Oligos have been commercially available for over 20 years at increasingly lower price points, a trend that is expected to continue [15]. At current time of writing, the price for 100 nanomole of a 11-nucleotide sequence is quoted at 6.60 USD by IDT, a leading vendor in synthetic DNA. Longer strands of DNA ($>$ 200nt) are obtained by ligating shorter oligos or obtained directly from plasmids of live organisms [12]. In the latter case, there is less control over the nucleotide sequence and length.

2.3 Domain-level abstraction and strand displacement

The interaction between strands of DNA is highly programmable through control of the strand sequence. In general, two strands do not hybridize unless parts of their sequences are complementary. An efficient way to reason about strand interaction is by assigning a shorthand for sequences and their complement. In Fig. 2.4b the top strand consists of three domains A, B and C. Their complementary domains are simply denoted with an asterisk modifier, so that the bottom strand of the duplex in the example consists of domains C* and B*. If we assume that the partial sequence overlap between non-complementary domains is sufficiently small, so that we can safely ignore the interaction between non-complementary domains, then a description of the possible hybridization reactions within a set of DNA strands is greatly simplified.

DNA strand displacement occurs when an incoming strand disrupts existing base-pairing of a duplex, so that one of the strands is replaced by the incoming strand. Toehold-mediated strand displacement occurs when a short toehold domain is used so that the invading strand initially docks into the duplex by hybridizing to the toehold, as in Fig. 2.5, and is a highly

[1]Such sequences suffer from lower yield and purity in the synthesis process. IDT website 'G Repeats. Structural Challenges for Oligo Design', accessed July 2015. `https://www.idtdna.com/pages/decoded/decoded-articles/core-concepts/decoded/2013/01/17/g-repeats-structural-challenges-for-oligo-design`

practical technique to create cascading DNA reaction networks. The toehold-mediated stand displacement reactions depicted in Fig. 2.5 are based on the operation of a 'see-saw' gate, which can act as a logic gate [16, 11, 17]. Toehold-mediated strand displacement was first used in a 'DNA-tweezer' to switch the device from an open to closed position [18].

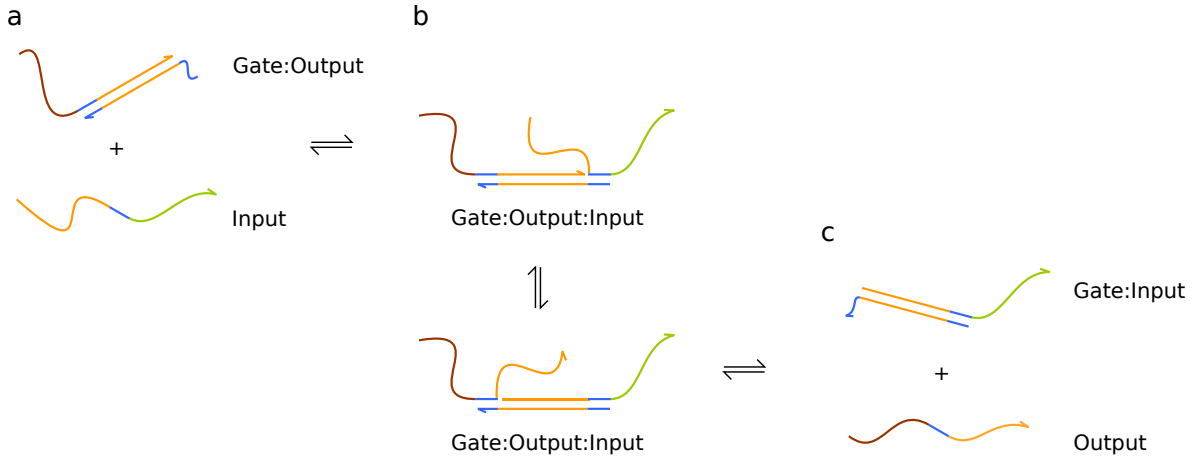


Figure 2.5: Toehold-mediated DNA stand displacement during the (partial) operation of a see-saw gate [16]. a) The gate-output duplex consists of two strands and one of the blue toehold domains is exposed. The input strand contains a domain that is complementary to the exposed toehold (also in blue). b) After binding to the toehold, the input strand displaces the orange domain of the output strand. c) The blue domain consists of only six nucleotides and the output strand spontaneously unbinds after the orange domain is displaced. The reverse reaction occurs at a roughly equal rate (for equal species concentrations) because the gate-input complex similarly has an exposed toehold domain.

2.4 DNA origami

DNA origami is a robust technique to create structures from DNA, and since its inception it has been employed to create a wide variety of shapes. Origami structures are created from a single-stranded ring of DNA, also called the scaffold, and the scaffold is typically several thousands of bases long. The scaffold is mixed in solution with an excess of shorter, linear single-stranded DNA, which are called staples. The solution is then cooled down from 95 °C to room temperature. During the cooling, the staples start binding to the scaffold through domain complementarity, and eventually the scaffold is fixed into an ensemble of tightly connected helices (see Fig. 2.6). Various shapes are possible by manipulating the nucleotide sequence of the staples, for example see the hexagon and star shapes in Fig. 2.7. The circular scaffold used in the original 2006 publication ([19]) was derived from the M13mp18 bacteriophage (virus), while the shorter staples were synthesized commercially. Many of the subsequent origami designs similarly use virus-derived scaffolds. DNA origami are regularly produced at nanomolar concentration.

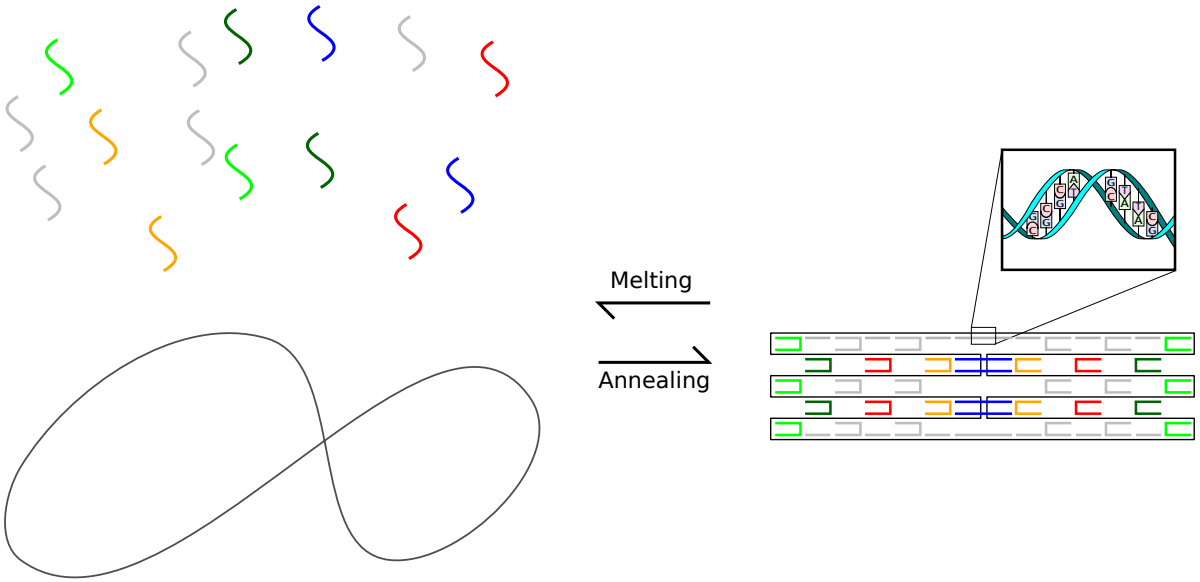


Figure 2.6: DNA origami is an ensemble of interwoven double-stranded DNA. The structure consists of one scaffold and many staple strands. In this example the staple strands are complementary to one or two domains on the scaffold.

At this concentration, a single drop contains in the order of 10^{10} origami.

The uniformity of origami objects can be assessed in part by agarose gel electrophoresis, where the sample is loaded into a well on one side and a voltage is applied between the two ends of the gel. A staining dye is used to reveal the position of the DNA after a fixed time. Because smaller fragments move through the gel faster, wells with many different bands indicate that the sample contains objects of varying size. Small defects in an origami, such as a missing staple, may not significantly impact its electrophoretic mobility.

Finally, we briefly discuss three methods for the imaging of DNA origamis.

- Atomic force microscopy (AFM): Mica is a silicate crystal that, after cleaving, produces highly flat surfaces. The negatively charged origamis stick to the similarly negatively charged mica surface in the presence of divalent cations (adsorption) [20]. After depositing the origami on the mica surface, an oscillating tip is positioned just above the mica surface. Differences in surface height and material between the surface and the tip are detected as changes in the amplitude of oscillation.

- Transmission electron microscopy (TEM): The scattering of electrons by DNA is relatively weak, leading to noisy images, while higher dosages of electron radiation leads to sample damage. To improve the contrast obtained from electron microscopy, the origami is treated with a staining dye, such as uranyl formate [21], which binds to the DNA.

- Cryo-electron microscopy (cryo-EM): After flash-freezing, the sample is imaged through

regular TEM without using staining dyes. Because the samples are fixed in vitreous ice, the nominally identical objects can be observed at multiple angles, enabling the reconstruction of a three-dimensional density model [22].

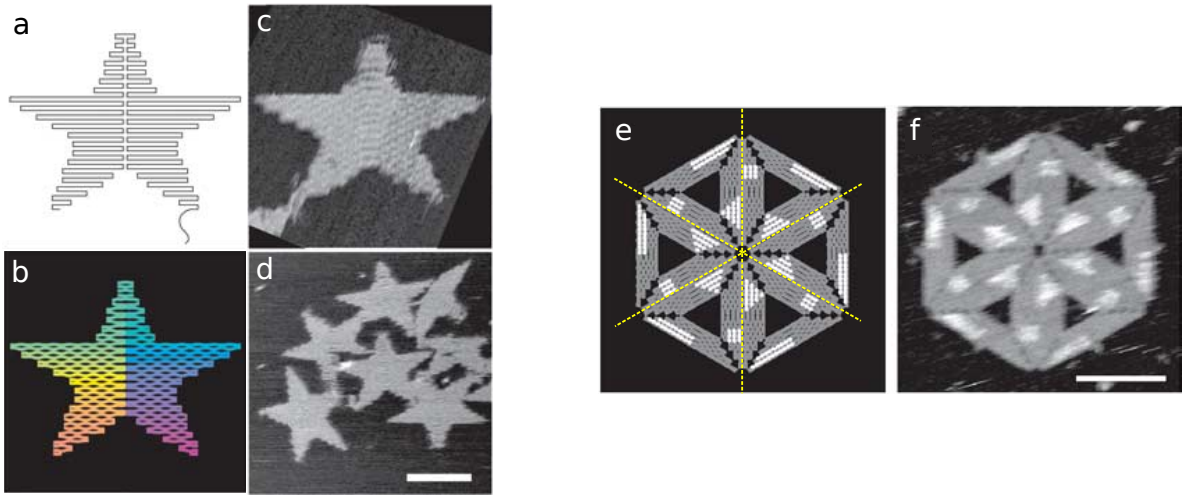


Figure 2.7: DNA origami in the shape of a star and a hexagon, reproduced from [19] with permission. a) The target shape of the scaffold routing resembles a star. b) Rendering of the target shape: with staple strands in place, the scaffold forms a mesh of double-stranded DNA. c,d) After preparation and deposition on a mica surface, the DNA origami is imaged using Atomic Force Microscopy (AFM, scalebar 100nm). e) In this design, designated staple strands mediate bonds between separate origami, creating a hexagon from six triangular origami. The surface is decorated with DNA hairpin loops. f) , and imaged using AFM (scalebar 100 nm). Note that (a,b,e) are renderings of the intended structure, they are results of AFM imaging.

2.5 Duplex formation

Simple two-state models for bimolecular reactions of complementary DNA strands are well established in the literature [23, 24, 25]. We recall definitions and results from statistical physics and their relation to DNA duplex formation. Given a fixed volume of buffer solution, let complementary strands A, B bind reversibly, forming the double-helical complex AB, as

$$A + B \rightleftharpoons AB. \tag{2.1}$$

The concentration of each species is denoted as $[A]$, $[B]$ and $[AB]$ which, in this thesis, we express in molar units M = mol/litre or nM = 10^{-9} mol/litre. Under the assumption of mass-action kinetics, the concentration of each species is given as

$$\frac{d[AB]}{dt} = k_+[A][B] - k_-[AB] \tag{2.2}$$

Figure 2.8: A simple duplex consisting of a three base-pair duplex (TCG/CGA) and a dangling strand (ATT). Under the nearest neighbour model, the two pairs of neighbouring base pairs are TC/GA and CG/CG.

for the binding and unbinding rate constants k_+, k_-. The equilibrium concentrations $\{A\}, \{B\}$ and $\{AB\}$ then follow

$$\frac{\{AB\}}{\{A\}\{B\}} = \frac{k_+}{k_-} = \exp\left(\frac{-\Delta G^{0\ \text{duplex}}_{AB}}{RT}\right) \times \text{M}^{-1} \tag{2.3}$$

where M^{-1} occurs to balance the units and R denotes the molar gas constant and T temperature. The sequence-specific Gibbs free energy change of duplex formation at molar concentration (1M) is given by

$$\Delta G^{0\ \text{duplex}}_{AB} = \Delta H^{0\ \text{duplex}}_{AB} - T\Delta S^{0\ \text{duplex}}_{AB} \tag{2.4}$$

where $\Delta H^{0\ \text{duplex}}_{AB}$ and $\Delta S^{0\ \text{duplex}}_{AB}$ are the sequence-specific enthalpy and entropy (also relative to molar concentration). For a given duplex, these quantities are estimated through the nearest-neighbour model of SantaLucia [24], in which $\Delta H^{0\ \text{duplex}}_{AB}$ and $\Delta S^{0\ \text{duplex}}_{AB}$ are assumed to be independent of temperature. A reduction of $\Delta G^{0\ \text{duplex}}_{AB}$ increases the ratio of k_+/k_- and increases the concentration $\{AB\}$ at equilibrium.

The melting temperature of the duplex is defined as the temperature where, assuming equilibrium, half the strands form a duplex, so that $\{A\} = \{B\} = \{AB\}$, assuming an equal initial concentration of A and B. Let the initial strand concentration be so that $[AB] = 0$ and $[A] = [B] = C/2$, where C is the initial concentration of strands, then the melting temperature is given by

$$T_M(AB) = \frac{\Delta H^{0\ \text{duplex}}_{AB}}{\Delta S^{0\ \text{duplex}}_{AB} + R\ln(C/4)}. \tag{2.5}$$

for non-selfcomplementary sequences [24].

Example 2.5.1. *Under the nearest-neighbour (NN) model we compute the enthalpy and entropy terms for a short duplex with a terminal dangle, as in Fig. 2.8, relative to the unhybridized state.*

The NN terms used here are accurate for duplexes ranging between duplexes ranging in length of four to sixteen basepairs [26], but for simplicity our example uses a duplex consisting of just three basepairs. The terms that arise in the model are:

$$\Delta G^0 = \Delta G^0_{\text{init}} + \Delta G^0_{\text{term.AT}} \tag{2.6}$$

$$+ \Delta G^0_{\text{TC}} + \Delta G^0_{\text{CG}} + \Delta G^0_{5'\text{AA}-\text{dangle}} \tag{2.7}$$

where ΔG^0_{init} is the initialisation term, $\Delta G^0_{\text{term.AT}}$ accounts for A/T base-pairs occurring at the end of the duplex, $\Delta G^0_{5'\text{AA}-\text{dangle}}$ is the (de)stabilisation term that accounts for the occurrence of the single stranded dangle at the 5' end of the duplex, and the other terms account for the neighbouring basepairs. The enthalpy and entropy terms for these terms, are estimated experimentally ([26], Table 1, [27] Table 2). Ignoring the contribution from salt dependencies, ΔS^{salt}, the enthalpy, entropy and free energy are:

$$\Delta H^0 = 0.2 + 2.2 - 8.2 - 10.6 - 0.5 \tag{2.8}$$

$$= -16.9 \ kcal \ / \ mol \tag{2.9}$$

$$\Delta S^0 = -5.7 + 6.9 - 22.2 - 27.2 - 1.1 \tag{2.10}$$

$$= -49.3 \ cal \ / \ mol \ K \tag{2.11}$$

where K is the SI unit for temperature (Kelvin) and where $R = 1.9872 \ cal \ mol^{-1}K^{-1}$. The free energy at $T = 37\,^\circ\text{C}$ is given by

$$\Delta G^0_{37\,^\circ\text{C}} = \Delta H^0 - T_{37\,^\circ\text{C}}\Delta S^0 \tag{2.12}$$

$$= -1.61 \ kcal \ / \ mol \tag{2.13}$$

Example 2.5.2. *Under the nearest-neighbour model we compute the enthalpy and entropy terms for a fully complementary 16-nt duplex, averaged over all possible sequences, to be*

$$\Delta H^{0 \ \text{duplex}} = -121.16 \ kcal \ mol^{-1} \tag{2.14}$$

$$\Delta S^{0 \ \text{duplex}} = -334.29 \ cal \ mol^{-1}K^{-1} \tag{2.15}$$

where K is the SI unit for temperature (Kelvin) and where $R = 1.9872 \ cal \ mol^{-1}K^{-1}$. The free energy at $T = 37\,^\circ\text{C}$ and the melting temperature of the hypothetical duplex (using $C = 20 \ nM$)

are given by

$$\Delta G^{0\ \text{duplex}}_{T=37\,^\circ\text{C}} = -17.48\ kcal\ mol^{-1} \tag{2.16}$$

$$T_M = 52.31\,^\circ\text{C} \tag{2.17}$$

Formation of the double-stranded helix enables hydrogen bonds and hydrophobic stacking interactions, and the change in enthalpy for the reaction, $\Delta H^{0\ \text{duplex}}_{AB}$, is negative. The difference in entropy is given by the difference in the number of available microstates before ($\Omega_{A,B}$) and after (Ω_{AB}) the reaction:

$$\Delta S^{0\ \text{duplex}}_{AB} = R \ln \left(\frac{\Omega_{AB}}{\Omega_{A,B}} \right) \tag{2.18}$$

The formation of the helix restricts the position and shape of the strands, reducing the number of possible microstates, that is, $\Omega_{AB} < \Omega_{A,B}$. As a result, $\Delta S^{0\ \text{duplex}}_{AB}$ is negative for the reaction. Overall, the formation of duplexes is favourable, and the melting temperature of $6-18$-nt long complementary strands is in the range of $30-60\,^\circ$C. Generally, longer duplexes are more stable than shorter ones, and strands with a higher C-G/A-T ratio are typically more stable than those with a lower C-G/A-T ratio.

2.6 Literature review

We now review literature on DNA nanotechnology.

Self-assembled structures

In 1982 Seeman proposed to use DNA latices to align proteins for the benefit of their structural analysis through X-ray crystallography [28]. An immobile junction by Seeman et al. was one of the first artificial nanostructures made out of DNA [29]. Other structures such as cubes, octahedra, tetrahedra and bipiramids have been created since [30, 31, 32, 33]. In 2006 Rothemund introduced new design paradigms to create nanostructures out of DNA, focused around the use of a large circular strand [19]. It is called 'origami' or folded DNA. Self-assembling structures of arbitrary shape can be made, for example smilies or stars were demonstrated in the original publication (also see Fig. 2.7). Hairpin loops positioned at the surface of the tile are useful to determine the orientation of the structure (see Fig. 2.7e,f), and, in one design, hairpin loops were used to embed a world map onto an origami tile [19]. Three-dimensional origami were subsequently developed. In one application containers with lids that open through the use of a strand-displacement reaction were created [34]. A similar construction was embedded with

anti-body fragments: when the box is locked, a cellular response is lacking, but when the box becomes unlocked, it exposes the fragments and a response occurs [35].

DNA origamis themselves can also self-assemble into larger structures. One obvious way would be to allow staples bind to two separate scaffolds, joining them. The stacking of origamis can also be achieved by creating a compatible geometry between objects [36, 37]. Because the end-to-end stacking of DNA helices is a stabilizing interaction, the geometrically compatible origamis self-assemble into stacked arrays. Some DNA objects are created without the use of a large backbone strand. One approach uses two-domain staples exclusively [38], and the staples are designed to form a solid tile or cube. Shapes are created by simply omitting sets of strands from the mix. This approach is called 'brick' or 'single-stranded tile' (SST) origami.

Software tools for self-assembled structures

The CADNANO software is a versatile tool to design origamis, and is developed by Shawn M. Douglas and co-workers [39]. Various views, including a three-dimensional rendering, helps the user to visualize and design origami structures. It contains functions to route scaffold in a predetermined shape and contains functionality to generate, manipulate and export nucleotide sequences. The software CANDO computes the mechanical stability of DNA structures through a finite-element method and supports import of CADNANO designs. CANDO is maintained by Mark Bathe and co-workers [40].

DNA-based molecular walkers

Dynein and Kinesin are families of natural protein that move along microtubials that span across the cell, hydrolysing ATP as fuel. Their function is to transport larger molecules across the cell and they perform certain roles during cell division. Synthetic molecular motors inspired by Dynein and Kinesin have been created. We now discuss the development of synthetic molecular motors made from DNA, which we call molecular walkers.

One example is that of Seeman et al. [41] who in 2004 report on a bipedal DNA walker. Each leg of the walker is a strand of DNA, and in the starting position, each leg is bound to a complementary strand, and three of such complementary strands form a small track. Each leg of the walker is locked and unlocked by adding input strands: one unlocking and one locking strand are required to move the first leg, and then two more strands are required to move the second leg. Another bipedal walker was demonstrated to walk across four of such complementary strands ('anchorages'), also using one input strand for each locking and unlocking reaction [42]. To traverse the track, nine sequential inputs are used over a period of 3 hours.

The walker that we consider in Chapter 4 is based on a design published in 2005: the study

demonstrates a single-legged walker to autonomously traverse a track of three anchorages [43]. This application uses a DNA nicking enzyme, that nicks the walker-anchorage complex and exposes a toehold domain on the walker strand after nicking. The walker then preferentially binds to the nearest intact anchorage, which is thermodynamically favourable because the number of hybridized nucleotides increases after stepping onto the next anchorage. Nicked anchorages no longer contain the domain complementary to the exposed walker toehold, preventing the walker from stepping onto nicked anchorages, resulting in directionality in the movement of the walker. The movement of the walker is conditional upon the presence of the nicking enzyme, and stepping occurs approximately once per minute. Later studies demonstrate movement across larger tracks, movement across junctions and detailed imaging of the stepping mechanism [44, 45], which we further discuss in Chapter 4. Other walker systems include [46, 47, 48, 49, 50].

DNA computation

Given the success of contemporary methods in molecular biology, and the discovery of molecular pathways that control cellular behavior, we might wonder what kind of computation can be implemented *in vitro* using biological molecules. It might be possible to create synthetic molecular computers that interact with biological tissue directly, opening up new possibilities for diagnostic and therapeutic devices. Or, such molecular computers might help us understand how computation works within the cell, through the process of reverse engineering. This area of research is called molecular computation. When we limit the scope of to computers made from DNA, we call it DNA computation. The limitation to devices made from DNA seems arbitrary, however, the cost and ease of manipulating biological molecules must also be considered. To review the advances made within DNA computation, we examine three computational models and their implementations.

Theoretical models of computation are as diverse as the application of computation. The decision process of an infantryman might be approximated well by a finite automaton, while a computer program might be best described using lambda-calculus, and central processing units are sometimes modelled as finite-memory Turing machines. We now describe computers, made from DNA, whose computational models are best described as parallel random search, tile assembly, and logic gates.

In 1994 Adleman [51] spurred synthetic molecular computation into reality by using DNA as a substrate to compute the solution to a directed Hamiltonian path problem. By using ligation enzymes to generate random paths over the directed graph, the existence of a solution is demonstrated if a path of an appropriate form is found. This was exciting for two reasons. Firstly, the problem is NP-complete, meaning any problem of this class could potentially be

solved using this approach. Secondly, Adleman created a massive parallel computation executed at small scale and low energy cost.

Adleman postulated that each ligation is a logical operation, and because a vial can easily contain 10^{13} strands of DNA, a high computational throughput is achieved. A similar approach, though employing hairpin DNA, was used in 2000 to compute a Boolean satisfiability problem over six variables, and in 2002 a 20-variable satisfiability problem was solved, which identified a solution from over a million possible combinations [52, 53]. Although a seminal contribution, the approach of Adleman did not translate to a truly viable method to solve routing problems, and silicon computers remain the most practical method to solve NP-hard problems.

Wang tiling [54] is a mathematical theory about squares with coloured sides, where squares can lie adjacent only if the sides match in colour. Any Turing machine can be embedded as a set of Wang tiles [55], and therefore the model is capable of universal computation. In 1996 Winfree proposed the use of Seeman's double-crossover (DX) molecules [56] as a substrate for computation, drawing inspiration from the theory of Wang tiles [57]. Winfree created a corresponding model of tile computation, the abstract tile assembly model (aTAM), and proved that this model is capable of Turing-universal computation [58].

DX tiles consist of four DNA strands that keep together through complementary hybridization. Each individual DX tile has four single-stranded domains, so that a tile can link with up to four other DX tiles, provided their domains are complementary. In 1998 Winfree et al. demonstrated that DX tiles self-assemble into large meshes [59]. In the same year Winfree formulated the kinetic tile assembly model to simulate the assembly of such meshes [60]. DX tiles were subsequently used to create a Sierpinski triangle and to implement a binary counter [61, 62].

DNA is an excellent substrate to implement chemical reaction networks [63], and we now discuss the development of these devices. Tweezers constructed from DNA were demonstrated in 2000 [18]. The appeal of this device (Fig. 2.9) is its ability to switch state when it interacts with certain DNA strands, so that the device responds to a signal (the incoming stands). In the closed state a quencher is in close proximity to a fluorophore, inhibiting fluorescent response of the fluorophore. In the open state the distance between the quencher and fluorophore increases and as a result the fluorescent response increases: the change in overall response is measured using a fluorometer. The initial signal strand is complementary to two separate domains in the device and after binding the device is in the closed state. In this closed state the signal strand is only partly hybridized and exposes a toehold domain. When a strand complementary to the signal strand binds to the toehold, it strips the signal strand from the device, causing the tweezers to 'open'. This means that waste, in the form of a DNA duplex, is produced after switching the device back to the open position. Because the 'waste' does not have any

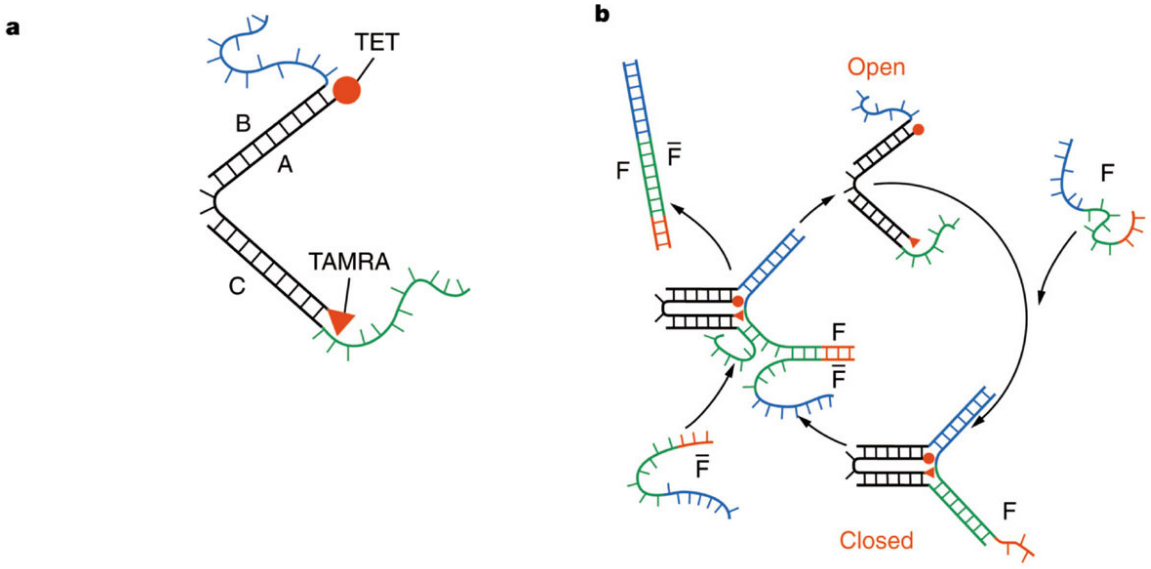


Figure 2.9: Reproduced from [18] with permission. DNA tweezers. a) Two strands are hybridized to strand A, leaving a short section single stranded between the two double stranded domains. A fluorophore (TET) and quencher (TAMRA) are attached to double stranded domains. b) Adding strand F to the solution causes the open tweezer to become closed. Strand F binds to two domains and brings the fluorophore and quencher in close proximity. In the closed state, the fluorescent response is reduced. A toehold domain of strand F is exposed in the closed state and enables a complementary strand $\overline{\text{F}}$, to easily attach and re-open the tweezers.

unhybridized sections, it is considered non-interfering in subsequent operation of the device.

An early DNA-based molecular network that implements logic gates was motivated by medical application: a simple circuit would sense the presence of mRNA, and release a medicine only if a set of mRNA are each detected [64]. Work by Seelig et al. (2006) demonstrates the implementation of logic gates and catalytic loops without the use of enzymes [65]. In 2007 Zhang et al. construct a catalytic gate using sacrificial fuel strands that amplify an incoming signal strand [66]. The see-saw gate, an adapted version of the catalytic gate, is proposed by Qian et al. in 2009 and is activated once the input crosses a threshold [16]. The see-saw gates can simulate logic gates, and in one instance a cascade was implemented that computes the square root of a 4-bit number [11]. A different gate architecture was later demonstrated to execute a consensus algorithm [12].

The detection of mRNA or DNA from a biological sample is a coveted application of DNA displacement networks. The use of a cascade circuit could potentially replace reverse transcription via real-time polymerase chain reaction in some cases, reducing logistical requirements and enabling cost-savings. For example, the device might detect the presence of mutated genes that indicate drug resistances in a pathogen. For this application the sensitivity, specificity and throughput time of DNA displacement cascades must be improved, which remains an active topic of research [67, 68, 69].

Modelling and verification for DNA computation

Process algebras and programming languages have been created to formally specify computational networks made from DNA [70, 71, 72, 73, 74], and their computational expressiveness has been studied [63, 75]. Because the reaction networks of such applications are complex, they are ideally designed with the aid of a computer and checked for correctness prior to implementation. Visual DSD is a software tool based on the DNA strand displacement (DSD) language by Cardelli and Phillips [71]. In this software tool the user specifies the initial species at the domain-level abstraction. The software then automatically generates the reaction network. If initial conditions are specified, the software simulates the execution of the network. Visual DSD exports to PRISM, enabling probabilistic model checking of DNA strand displacement networks. In one instance this revealed a bug in the design of a logic gate [76]. Visual DSD was used during the development of the 4-bit square root circuit created by Qian et al. [11]. A similar approach to the analysis and verification of DNA reaction networks, although currently without a publicly available implementation, was subsequently proposed by Winfree et al. [74]. This approach is more permissive than the original language by Cardelli and Phillips, for example four-way branch migration reactions and branched structures are allowed. Similar to DSD, a separation of timescales (first-order reactions versus second order reactions) is used to simplify the reaction network. A recently published work by Phillips et al. extends the DSD language to allow similarly branched structures [77].

Modelling for DNA nanotechnology

Statistical descriptions of DNA stability exist in the form of nearest-neighbour models that account for neighbour pairs, dangling ends, nucleotide mismatches and bulge loop formation [24, 27, 78, 26, 79]. MFOLD and NUPACK are software for secondary structure prediction of DNA [80, 81], and are based on nearest neighbour models. Secondary structure prediction for DNA is difficult because of the vast number of combinations, and specialized algorithms are applied to find the configuration(s) that minimize free energy. The kinetics of domain migration and the influence of nucleotide sequence, including nucleotide mismatches, has attracted attention from both theoretical and experimental researchers [82, 83, 84, 85].

OxDNA is a simulation software for DNA and RNA based on a coarse-grained simulation of molecular dynamics that was proposed by Ouldridge et al. [86]. The ‘coarse’ abstraction makes OxDNA suitable to simulate nanoscale devices over timescales that are orders of magnitude larger than those obtainable through atomistic molecular dynamics. Applications of OxDNA include the simulation of Holiday junctions, DNA tweezers, displacement reactions and a DNA walker [84, 87, 88, 86, 89]. An atomistic simulation of a full-scale DNA origami was recently

employed to study the distribution of cations within the structure [90].

Comparison between DNA, RNA and polypeptides

We briefly compare DNA to RNA and protein as an alternative nanoscale engineering material. Predicting protein structure from peptide sequence alone is notoriously difficult and, although it is possible to design simple synthetic proteins, this typically requires simulation-based design software to predict the eventual shape [91, 92]. In contrast, DNA enables robust manufacturing of structures without the use of pre-production molecular dynamic simulations. In one approach principles of self-assembly were inferred from DNA-based devices and applied to create a synthetic protein that assembles into a tetrahedron [93], and in another nanoscale sheets of protein were created [94]. Synthetic structures made from folded RNA were recently demonstrated [95], where the structure consists of a single nucleotide chain and folds through self-interaction. As the applications of self-assembled nanoscale engineering develop, it seems likely that these fields will continue to inspire each other.

Chapter 3

Introduction to probabilistic modelling and verification

In this chapter we review the main concepts relevant to probabilistic model checking that feature in the thesis. From Section 3.3 on we discuss relevant literature.

Probabilistic model checking aims to, given a probabilistic model, automatically verify statements such as:

The system resolves conflict within 1.0 seconds, with a probability of at least 99.999%.

Using a probabilistic model checking software, the user, after constructing the model, only has to specify the statement in a temporal logic, and a model checking algorithm validates if the statement holds or not. In this way, the user does not have to make manual adjustments to their model, in order to compute the desired properties. The property specification in this case might look like:

$$\mathtt{P}_{\leq 99.999\%}\left[\ \mathtt{conflict}\ \ \mathtt{U}^{\leq 1.0}\ \mathtt{resolved}\ \right] \tag{3.1}$$

In other words, when in a state of conflict, the system reaches a resolve within 1.0 second, with at least 99.999% probability. Probabilistic model checking also attempts to answer the question of *which set of properties* can be answered, and with which computational complexity.

In the model checking portion of this thesis, we demonstrate how to synthesize model parameters such that a given query holds (Section 5.2.3), and we demonstrate how to compute reward properties for an improved version of the uniformisation method (Section 5.2).

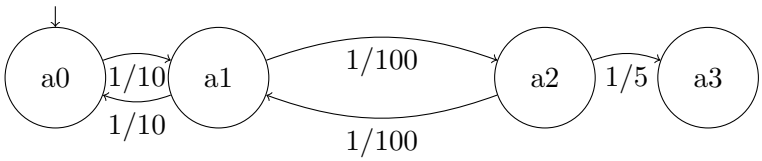


Figure 3.1: Continuous-time Markov chain model of a molecular walker. In this process, the initial position (state) of the walker is $a0$. The delay in movement from $a0$ to $a1$ is given by an exponential distribution, Exp(1/10), so that the average waiting time before the walker moves is equal to $\frac{1}{1/10} = 10s$. From position $a1$, the walker moves back to $a0$ with probability $\frac{10}{11}$ and moves to $a2$ with probability $\frac{1}{11}$.

3.1 Markov chains

Continuous-time Markov chains are used in this thesis to model a DNA walker and the self-assembly process of DNA origami, and to enable the probabilistic and statistical model checking of these models. We now describe these Markov chains and methods to analyse them. The definitions of continuous-time Markov chains and their generating matrices (Def. 3.1.1, Def. 3.1.3 and Def. 3.1.7) and the definitions of continuous stochastic logic (Def. 3.2.1 and Def. 3.2.4) are based on [96, 97].

Definition 3.1.1 (Continuous-time Markov chain (CTMC))**.** *A CTMC is a tuple* $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ *where* S *is a countable set of states,* $\pi_0 : S \to [0,1]$ *is the initial state distribution with* $\sum_{s \in S} \pi_0(s) = 1$*,* $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ *is the rate matrix and* $L : S \to 2^{AP}$ *is a labelling function for a set of atomic propositions* AP*. A transition between states* $s, s' \in S$ *can occur only if* $\mathbf{R}(s, s') > 0$ *and in that case the probability of triggering the transition within* t *time units equals* $1 - e^{-t\mathbf{R}(s,s')}$*. The time spent in state* s*, before a transition is trigged, is exponentially distributed with exit rate* $E(s) = \sum_{s' \in S} \mathbf{R}(s, s')$*, and when the transition occurs the probability of moving to state* s' *is given by* $\frac{\mathbf{R}(s,s')}{E(s)}$*.*

A graph-based representation is the preferred method of depicting CTMCs that consist of a limited number of states, such as in Fig. 3.1. The initial state is marked with an incoming arrow. Each transition is depicted as a directed edge that is labelled with the transition rate. In this case $a0$ is the initial state, and the exit rate for that state is given by $E(a0) = 1/10$. The time until the Markov process leaves state $a0$ is exponentially distributed with parameter $\lambda = \frac{1}{10}$ and the expected time until a transition occurs is given by $\frac{1}{\lambda} = 10$. After the first transition, the process necessarily ends up in state $a1$. The expected waiting time in state $a1$ is given by 100/11. States that have two or more exit rates are affected by the so-called race condition, where the probability of triggering each transition is directly proportional to the rate of the transition. The probability of moving to state $a2$ from state $a1$ is 1/11, while the probability of moving to $a0$ is 10/11.

Example 3.1.2 (Molecular walker)**.** *The Markov chain in Fig. 3.1 models a molecular walker, which moves randomly due to thermal noise. Starting in location $a0$, it moves between location $a0$ and location $a1$. From location $a1$, the walker can bridge a gap and move to location $a2$, although movement across the gap occurs at a lower rate. From location $a2$ the walker can access the state $a3$, where the walker becomes trapped. Randomly generated paths (generated by the algorithm in Def. 3.1.12) over the CTMC are given in Table 3.1.*

Definition 3.1.3 (Paths over continuous-time Markov chains)**.** *A path ω over a CTMC $(S, \pi_0, \mathbf{R}, L)$ is an alternating sequence of states s_i and time delays t_i for which $\boldsymbol{R}(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{\geq 0}$ for $i \geq 0$. The value t_i represents the amount of time spent in state s_i. Denote the set of paths starting in $s \in S$ by Path(s). The path ω also defines a path function $\omega : \mathbb{N} \to S$ such that the i-th state in the path is given by $\omega(i) = s_i$. The state at time t for path ω is denoted as $\omega@t$, and is equal to $\omega(i)$ for the smallest i such that $\sum_{n=0}^{i} t_n \geq t$. We also define a function time such that $time(\omega, i) = t_i$ if it exists.*

Definition 3.1.4 (Probability measure over paths in a CTMC)**.** *A probability measure over paths of a Markov process over a CTMC $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ is given by the cylinder construction and we follow [97]. Given a state s and a fixed number of jumps k, fix k non-empty intervals I_i of $\mathbb{R}_{\geq 0}$ and visited states $s_i \in S$ for $0 \leq i < k$ where $\mathbf{R}(s, s_0) > 0$ and $\mathbf{R}(s_i, s_{i+1}) > 0$ for $0 \leq i < k-1$ and let the cylinder set $C(s, I_0, s_0, \ldots, I_{k-1}, s_{k-1})$ be given as*

$$\forall \omega \in C(s, I_0, s_0, \ldots, I_{k-1}, s_{i-1}) \forall j \in 0..k-1 : time(\omega, i) \in I_j \text{ and } \omega(i) = s_i \tag{3.2}$$

Let $\mathcal{F}$ be the smallest σ-algebra containing cylinder sets for all $s \in S$ and k. The unique probability measure P over $\mathcal{F}$ is inductively defined as

$$P\left(C(s)\right) = \pi_0(s) \tag{3.3}$$

and

$$\begin{aligned} &P\left(C(s', I_0, s_0, \ldots, I_{k-1}, s_{k-1}, I_k, s_k)\right) = \\ &\quad P\left(C(s', I_0, s_0, \ldots, I_{k-1}, s_{k-1})\right) \frac{\mathbf{R}(s_{k-1}, s_k)}{E(s_{k-1})} (e^{-E(s_{k-1}) \inf I} - e^{-E(s_{k-1}) \sup I}). \end{aligned} \tag{3.4}$$

Continuous-time Markov chains are well suited to model real-time discrete stochastic processes, for example, the arrival of customers in a queue, or nuclear decay. Crucially, the behaviour of the model only depends on the current state, and is independent of previously visited states. Some stochastic processes are best described in discrete time, for example the evolution of a sports tournament is best expressed in rounds rather than explicit time. In that case a model

Path	States visited	Waiting times								
ω_1	$a0, a1, a0, a1, a0, a1, a0, a1, a0, a1$	43.1	2.8	1.2	19.3	4.6	0.5	0.6	11.7	1.6
ω_2	$a0, a1, a0, a1, a2, a3$	4.8	5.7	0.7	6.6	2.5				
ω_3	$a0, a1, a0, a1, a0, a1, a0, a1, a2, a3$	6.7	2.0	2.2	7.0	7.1	21.1	5.6	10.5	15.1
ω_4	$a0, a1, a0, a1, a0, a1, a0, a1, a0, a1$	3.9	6.9	5.2	6.2	5.0	6.2	4.2	4.7	5.9

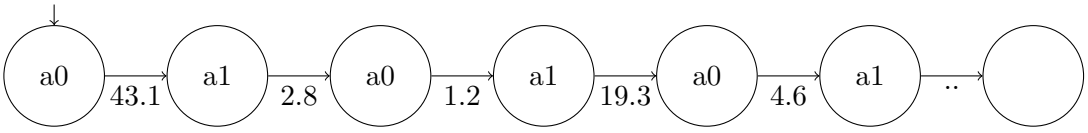


Table 3.1: Random movement by the molecular walker from Fig. 3.1 is simulated using the stochastic simulation algorithm (Def. 3.1.12). Top: For five independently simulated trajectories, the order in which the walker visits each location (state) is listed, as are the waiting times in each location. Bottom: The walker, given the simulated trajectory ω_1, bounces back and forth between location $a0$ and $a1$. The arrows between the states indicate the waiting time, in seconds, at each location.

based on a discrete-time Markov chain (DTMC) is appropriate. The notion of DTMCs arises naturally in the analysis of CTMCs and many other settings.

Definition 3.1.5 (Discrete-time Markov chain (DTMC))**.** *A DTMC is a tuple* $\mathcal{D} = (S, \tau_0, \mathbf{P}, L)$ *where* S *is a countable set of states,* $\tau_0 : S \to [0,1]$ *is the initial state distribution where* $\sum_{s \in S} \tau_0(s) = 1$*,* $\mathbf{P} : S \times S \to [0,1]$ *is the transition matrix such that* $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ *and* $L : S \to 2^{AP}$ *is a labelling function for a set of atomic propositions* AP*. During each time step the process moves from state* s *to* s' *with probability* $\mathbf{P}(s, s')$*.*

3.1.1 Uniformisation

We now derive the method of uniformisation, a frequently used method in probabilistic model checking. Uniformisation is a method of numerical integration, and other approaches, such as Runge-Kutta methods, can be used instead. However, uniformisation compares favorably to Runge-Kutta (RKF45) when a high precision is required [98]. In probabilistic model checking, a high precision is typically desired, for example, a typical statement resulting from an analysis would be:

The randomized protocol resolves conflict within 1.0 seconds in at least 99.999% of the cases.

In this case, a low tolerance for error is required. Note that absolute error bounds are straightforward to compute for the uniformisation methods. To introduce the method, consider the following example, where we derive a DTMC model $\mathcal{D}$ from a CTMC model $\mathcal{C}$ by uniformizing the transition rates:

Example 3.1.6 (Discrete-time walker)**.** *A DTMC model* $\mathcal{D} = (S, \tau_0, \mathbf{P}, L)$ *of the molecular walker, shown in Fig. 3.3, is derived from the continuous-time version in Fig. 3.1 by copying the*

state space S and initial distribution $\tau_0 = \pi_0$ from the CTMC and setting the transition matrix as

$$\mathbf{P}(s,s') = \begin{cases} \frac{1}{q}\mathbf{R}(s,s') & \text{if } s \neq s' \\ 1 - \frac{1}{q}\sum_{s' \neq s}\mathbf{R}(s,s') & \text{if } s = s' \end{cases} \tag{3.5}$$

where $\frac{1}{q} = 2$. Generating a DTMC in this way is called uniformisation: it is equivalent to adding a self-loop in each state s of the CTMC such that the exit rate is given as $E(s) = q$ and constructing the DTMC as $\mathbf{P}(s,s') = \frac{1}{q}\mathbf{R}(s,s')$. The uniformised DTMC retains the correct ratio of transition probabilities in each state: the probability of visiting states in a given order is equal between the DTMC and the CTMC.

We can now describe the vector of state probabilities at time t, which we denote by π_t, as the solution to a linear ordinary differential equation (ODE):

Definition 3.1.7 (Generating matrix, uniformised matrix and the uniformisation rate)**.** *For a CTMC $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$, let $\mathbf{E}$ be a $S \times S$ diagonal matrix such that $\mathbf{E}(s_i, s_i) = E(s_i) = \sum_{s' \in S}\mathbf{R}(s_i, s')$. The generating matrix $\mathbf{Q}$ is given by $\mathbf{Q} = \mathbf{R} - \mathbf{E}$. Then the vector $\pi_t : S \to [0,1]$ of the state probabilities at time t is given by $\frac{d\pi_t}{dt} = \pi_t\mathbf{Q}$ (linear ODE) and initial condition π_0 at $t = 0$. The uniformised matrix is given as $\mathbf{P} = \mathbf{I} + \frac{1}{q}\mathbf{Q}$, where $q \geq \max_s\{E(s) - \mathbf{R}(s,s)\}$ is called the uniformisation rate.*

Given a CTMC $\mathcal{C}$ there are many methods of solving the differential equation of Def. 3.1.7 for π_t, the vector of state-probabilities at time t. We now describe the uniformisation method, which derives the transient probabilities π_t seemingly straightforwardly by power expansion [99]:

$$\begin{aligned} \pi_t &= \pi_0 e^{\mathbf{Q}t} \\ &= \pi_0 \Sigma_{i=0}^{\infty}(\mathbf{Q}t)^i/i! \end{aligned} \tag{3.6}$$

Matrix $\mathbf{Q}$, however, contains both positive and negative entries. In practice, the transformation $\mathbf{Q} = q\mathbf{P} - q\mathbf{I}$ is used to rewrite Eq. 3.6:

$$\pi_t = \pi_0 e^{(q\mathbf{P} - q\mathbf{I})t} \tag{3.7}$$

$$= \pi_0 e^{-qt} e^{qt\mathbf{P}} \tag{3.8}$$

$$= \pi_0 e^{-qt} \Sigma_{i=0}^{\infty}(qt\mathbf{P})^i/i! \tag{3.9}$$

$$= \pi_0 \Sigma_{i=0}^{\infty} e^{-qt}\frac{(qt)^i}{i!}\mathbf{P}^i \tag{3.10}$$

The form of Eq. 3.10 is referred to as standard uniformisation [99, 100, 98]. As matrix $\mathbf{P}$ is a

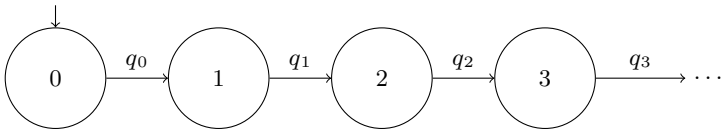


Figure 3.2: Birth process. The waiting time before the i-th jump ($i \geq 0$) is exponentially distributed with rate q_i.

stochastic matrix and contains only non-negative values, this approach is favoured over the form of Eq. 3.6 for enhanced numerical stability. The full form of standard uniformisation is derived from Eq. 3.10 by recognizing the term for Poisson probabilities. We now define the Poisson process, itself a special type of (pure) birth process.

Definition 3.1.8 (Birth process)**.** *The birth process is a CTMC where each state $S = \{0, 1, 2 \ldots\}$ enumerates the number of occurred transitions, as depicted in Fig. 3.2. The process starts from zero, $\pi_0(0) = 1$, and the delay between each transition is exponentially distributed with rate $q_i \in \mathbb{R}_{\geq 0}$ for i non-negative integer, that is*

$$\mathbf{R}(i,j) = \begin{cases} q_i & \text{if } j = i+1 \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

Definition 3.1.9 (Poisson process)**.** *The Poisson process is a birth process where the delay between each transition is exponentially distributed with equal rate, so that $\forall i : q_i = q$, $q \in \mathbb{R}_{\geq 0}$. The probability to be in state i by time t is given by $\gamma_{i,qt} = e^{-qt}\frac{(qt)^i}{i!}$.*

Definition 3.1.10 (Standard uniformisation (also: Jensen's method, randomisation))**.** *The transient state probabilities by time t are obtained by standard uniformisation as a sum of state distributions after i discrete-stochastic steps, weighted by the probability of observing i jumps in a Poisson process. The approximation $\hat{\pi}_t$ and the transient probabilities π_t are given as*

$$\pi_t = \sum_{i=0}^{\infty} \gamma_{i,qt}\tau_i \tag{3.12}$$

$$\hat{\pi}_t = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt}\tau_i \tag{3.13}$$

where the discrete-time probabilities are given as $\tau_i = \pi_0\mathbf{P}^i$, $\gamma_{i,qt} = e^{-qt}\frac{(qt)^i}{i!}$ denotes the i-th Poisson probability for a process with parameter qt, and k_ϵ satisfies the convergence bound $\sum_0^{k_\epsilon} \gamma_{i,qt} \geq 1 - \epsilon$ for fixed $\epsilon > 0$. The Poisson terms and summation bound are computed efficiently using an algorithm due to Fox and Glynn [101].

Example 3.1.11 (Transient probabilities for the molecular walker)**.** *We graphically show the*

uniformisation method applied to the example of the molecular walker. We compute π_{20}*, the vector of probabilities for the walker to be in each location at time* $t = 20$ *using the uniformisation rate* $q = \frac{1}{2}$*. The derived DTMC* $\mathcal{D} = (S, \tau_0, \mathbf{P}, L)$ *and initial discrete distribution* $\tau_0 = \pi_0$ *are depicted in Fig. 3.3. The discrete time process evolves as* $\tau_i = \tau_{i-1}\mathbf{P}$ *for* $i > 0$ *and* π_{20} *is found according to Definition 3.1.10 (dashed box, Fig. 3.3).*

Numerical solvers such as Runge-Kutta schemes are also suitable to integrate the differential equation $\frac{d\pi_t}{dt} = \pi_t\mathbf{Q}$, although uniformisation is preferred for its increased performance when a higher accuracy is desired [98]. In either case the computational complexity scales linearly in the size of S. The state space S itself potentially suffers from the problem of state space explosion, that is, the combinatorial blow up that occurs when the state space is exhaustive over many variables. Due to state space explosion, the cost of solving the differential equation can be prohibitive, and the solution vector π_t itself may not fit in main memory, let alone permit operations.

The stochastic simulation algorithm (SSA) enables the simulation of the Markov process and does not require each state in S to be stored in memory. The SSA provides an estimate to π_t, rather than an arbitrarily precise solution, and is attributed to Gillespie, who proposed it in the context of chemical reaction networks [102].

Definition 3.1.12 (Stochastic simulation algorithm)**.** *A path* ω *starting in* π_0 *at time* $t = 0$ *in CTMC* $(S, \pi_0, \mathbf{R}, L)$ *is simulated until time* T*. Assume the current time is* $t < T$ *and the current state is* s_i*. A single iteration of the algorithm is as follows:*

1) *Compute the exit rate* $E(s_i) = \sum_j \mathbf{R}(s_i, s_j)$ *and generate independent standard uniform random variables* $U_1, U_2 \sim U(0, 1)$.
   If $E(s_i) = 0$*, terminate the algorithm. Otherwise, the waiting time is given by* $t_i = \frac{-1}{E(s_i)} \log U_1$.

2) *If* $t + t_i > T$*, terminate the algorithm. Otherwise, increment* t *by* t_i *and select the next state:*
   Let $k = \min_n \sum_{j=0}^{n} \mathbf{R}(s_i, s_j) \geq U_2 E(s)$*. Move to state* s_k.

Some CTMCs can trigger an infinite number of transitions in only finite amount of time with non-zero probability, which is called Zeno behaviour. A sufficient condition against this behaviour is the assumption of bounded transition rates, i.e. $\sum_{s' \in S} \mathbf{R}(s, s')$ is convergent $\forall s \in S$ [97]. The CTMCs that we discuss in this thesis do not exhibit Zeno behaviour. Inhomogeneous CTMCs are CTMCs for which the rate matrix $\mathbf{R}$ depends on some parameter, and we discuss the time-inhomogeneous variant.

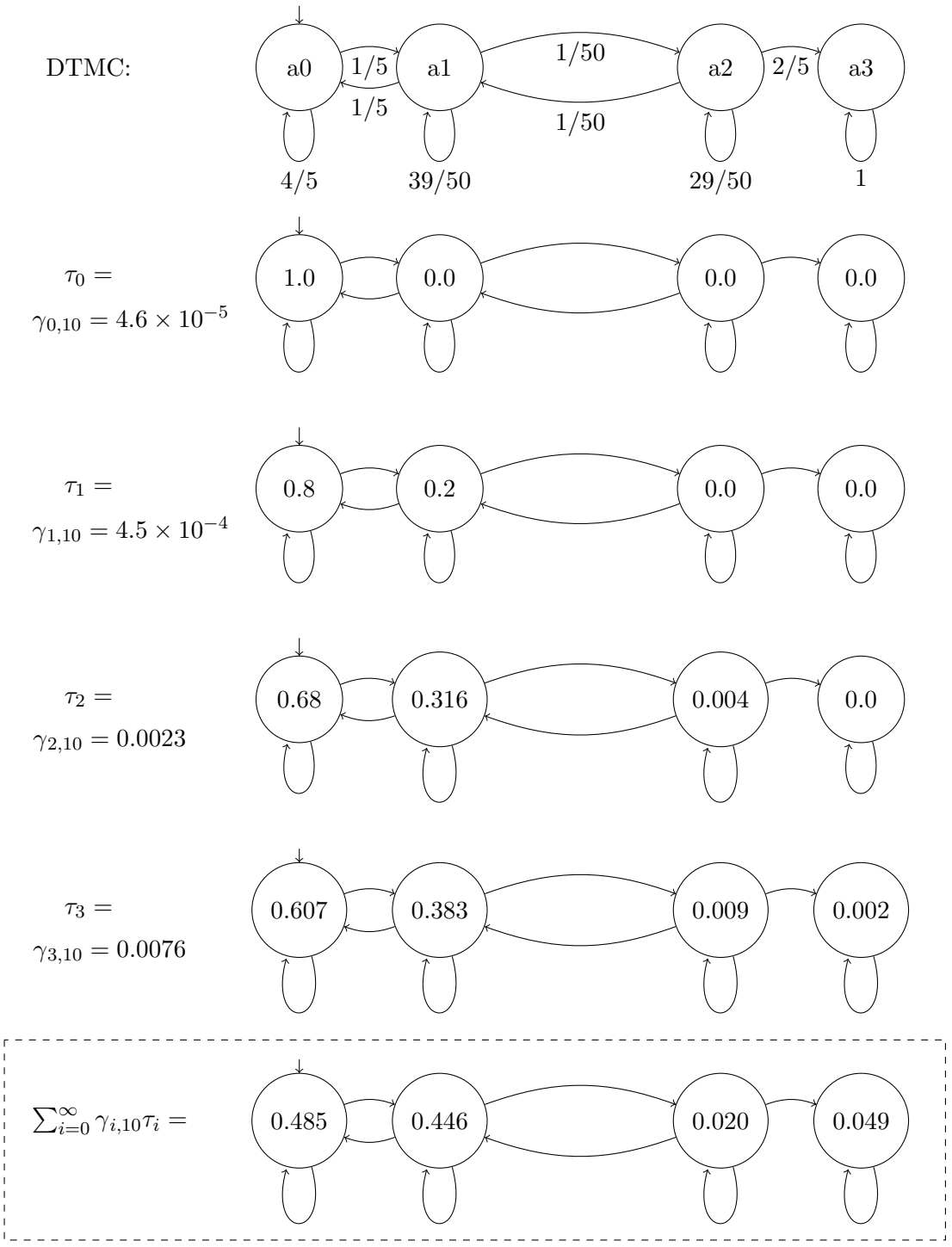


Figure 3.3: Discrete-time Markov chain model of the molecular walker (Fig. 3.1) obtained through uniformisation ($q = \frac{1}{2}$) and the discrete transient probability distribution τ_i of the DTMC model at iteration $i = 0, 1, 2, 3$. Dashed: The transient probabilities π_t for the CTMC at time $t = 20$ are obtained via uniformisation as $\pi_{20} = \sum_{i=0}^{\infty} \gamma_{i,10}\tau_i$ where γ are Poisson probabilities.

Definition 3.1.13 (Inhomogeneous continuous-time Markov chain (ICTMC))**.** *An inhomogeneous continuous-time Markov chain (ICTMC) is a tuple* $(S, \pi_0, \boldsymbol{R}_t, L)$ *where* S *is a set of states,* $\boldsymbol{R}_t : S \times S \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ *is a transition rate matrix and* $L : S \to 2^{AP}$ *a labelling function that maps each state* $s \in S$ *to a set of atomic propositions (AP). Although similar to the CTMC of Def. 3.1.1,* $\boldsymbol{R}_t$ *is now time dependent. The exit rate is given by* $E_t(s) = \sum_{s' \in S} \boldsymbol{R}_t(s, s')$*. The probability to remain for at least time* $t \in \mathbb{R}_{\geq 0}$ *in state* s *is equal to* $e^{-\int_0^t E_u(s)du}$*. The probability of moving to a state* $s' \in S$ *given a transition at time* t *is equal to* $\frac{\int_0^t \boldsymbol{R}_u(s,s')du}{\int_0^t E_u(s,s')du}$*.*

Provided the rate matrix is piecewise constant over a finite interval $[0, T] \subset \mathbb{R}$, i.e. $\mathbf{R}_t = \mathbf{R}_{t_k}$ for all $t \in [t_k, t_{k+1})$ where $t_k < t_{k+1}$ and $[0, T] = \cup_{k=0}^{n}[t_k, t_{k+1}]$, the transient probabilities are obtained in a piecewise fashion by solving a homogeneous CTMC by standard uniformisation with rate matrix R_{t_k} on each interval (Lemma 5.1.2, also Theorem 1 in [103]). Any ICTMC can be made piecewise constant by fixing rates for a set amount of time, thus approximating the original ICTMC to some degree. It should be noted that other numerical integration techniques, such as Runge-Kutta schemes, allow the direct computation of transient probabilities for inhomogeneous rates: no piecewise approximation is necessary. Uniformisation can be adapted to enable the analysis of inhomogeneous CTMCs [104]. In this thesis, stochastic simulation is the preferred method of analysis for inhomogeneous CTMCs (which we use to model the self-assembly of DNA origami in Chapter 6 and Chapter 7).

3.2 Continuous stochastic logic

The prevalent logic to describe properties for CTMC models is that of Continuous Stochastic Logic (CSL), an extension of Computation Tree Logic (CTL) [105]. CSL enables three types of property model checking, which we summarise as:

- Time-bounded properties that reason about behaviour up until a time T. For example, $\mathtt{P}_{=?}[A\ \mathtt{U}^{[0,T]}B]$ is equal to the probability of B becoming true before or at time T, while A is true until B is true. Also see Fig 3.4.
- Time-unbounded properties that reason about unbounded behaviour. For example, $\mathtt{P}_{=?}[\mathtt{F}(A \wedge B)]$, which is equal to the probability of eventually being in a state where A and B are true.
- Properties that reason about steady-state behaviour. For example, $\mathtt{S}_{\leq 0.2}[\mathtt{P}_{\geq 0.9}[\mathtt{F}^{[0,T]}A]]$ specifies that in the steady-state, the probability of satisfying the inner property, $\mathtt{P}_{\geq 0.9}[\mathtt{F}^{[0,T]}A]$, is less than 20%. The inner property specifies that the probability of reaching a state where A holds true within T time unit is at least 90%.

In this thesis, we only consider time-bounded properties, i.e., we are interested in how quickly

the walker reaches a final anchorage or how fast the DNA origami assembles. We now recall the standard definition of CSL (also see [105]):

Definition 3.2.1 (Continuous Stochastic Logic (CSL))**.** *The syntax of time-bounded CSL is as follows:*

$$\Phi := \texttt{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \texttt{P}_{\sim p}[\phi] \tag{3.14}$$

$$\phi := \texttt{X}\ \Phi \mid \Phi\ \texttt{U}^{I}\ \Phi \tag{3.15}$$

where a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, I is a finite interval of $\mathbb{R}_{\geq 0}$ and $p \in [0,1]$ is the probability threshold. $\texttt{P}_{\sim p}$ is the probabilistic operator and ϕ is a path formula. Given a CTMC $(S, \pi_0, \boldsymbol{R}, L)$ and a state $s \in S$, the relation $s \models \Phi$ is defined as:

$$s \models \texttt{true} \qquad \forall s \in S \tag{3.16}$$

$$s \models a \qquad \Leftrightarrow\ a \in L(s) \tag{3.17}$$

$$s \models \neg\Phi \qquad \Leftrightarrow\ s \not\models \Phi \tag{3.18}$$

$$s \models \Phi_1 \wedge \Phi_2 \qquad \Leftrightarrow\ s \models \Phi_1 \wedge s \models \Phi_2 \tag{3.19}$$

$$s \models \texttt{P}_{\sim p}[\phi] \qquad \Leftrightarrow\ P(\ \omega \models \phi \,|\, \omega \in\ Path(s)\ ) \sim p \tag{3.20}$$

where the path formula is expanded as

$$\omega \models \texttt{X}\Phi \qquad \Leftrightarrow \omega(1) \text{ is defined and } \omega(1) \models \Phi \tag{3.21}$$

$$\omega \models \Phi_1\ \texttt{U}^{I}\ \Phi_2 \qquad \Leftrightarrow \exists t \in I\ s.t.\ [\omega@t \models \Phi_2\ \wedge\ \forall r \in [0,t)\ :\ \omega(r) \models \Phi_1)] \tag{3.22}$$

A distribution $\pi : S \to [0,1]$ is said to satisfy a CSL state formula $\pi \models \texttt{P}_{\sim p}[\psi]$ by fair weighting of the satisfaction probability:

$$\pi \models \texttt{P}_{\sim p}[\psi] \Leftrightarrow\ \left(\sum_{s \in S} \pi(s) P(\ \omega \models \phi \,|\, \omega \in\ Path(s)\ )\right) \sim p \tag{3.23}$$

In addition, $\sim p$ can be replaced by $=?$, indicating the probability of satisfaction, instead of satisfiability. Additional abbreviated notation is listed in Table 3.2. We now give examples of time-bounded CSL properties.

- *The time-bounded until property*: $\texttt{P}_{=?}[A\ \texttt{U}^{[0,t]}\ B]$. The probability of B becomes true by time t and A is true until B becomes true.
- *A nested property:* $\texttt{P}_{\geq 0.9}[A\ \texttt{U}^{[0,t]}\ \texttt{P}_{\geq 0.8}[B\ \texttt{U}^{[0,t]}\ C]]$. Let S_{nested} be the set of states for which the probability of satisfying B before eventually satisfying C within t time units is at least

80%. Then this property describes that the probability of satisfying A before eventually reaching a state in S_{nested} within t time units is at least 90%.

The problem definition of CSL model checking for CTMCs is as follows.

Definition 3.2.2 (CSL model checking)**.** *Given a CSL specification Φ, CTMC $\mathcal{C}$, and state $s \in S$, decide if the Markov process starting in s satisfies Φ, that is*

$$s \models \Phi \tag{3.24}$$

Model checking a time-bounded CSL formula is in practice reduced to time-bounded reachability of a set of target states [97], and we offer a description for the property $\mathtt{P}_{=?}[A \ \mathtt{U}^{[0,t]} \ B]$, also illustrated in Fig. 3.4.

Example 3.2.3 (Time-bounded CSL model checking)**.** *Given the property*

$$\mathtt{P}_{=?}[A \ \mathtt{U}^{[0,t]} \ B] \tag{3.25}$$

and CTMC $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$, denote the derived CTMC $\mathcal{C}' = (S', \pi_0, \mathbf{R}', L)$ by modifying the state space S and transition matrix $\mathbf{R}$ in the original CTMC $\mathcal{C}$. The set of target states T is given by the states that satisfy B; define $T = \{s \models B\}$, and to satisfy the property the process must reach a state in T by time t. In the modified CTMC, all outgoing transitions are removed from the set of target states, e.g., $\forall s \in T : \mathbf{R}'(s, s') = 0$. Then in the set of remaining states $S \setminus T$ the transitions from states satisfying A into states that satisfy $\neg A$ are replaced by a transition into a new deadlock state, s_{DEADLOCK}. All transitions from states $s \in S \setminus T$ that do not satisfy A are removed from the model. Summarizing, for states $s, s' \in S$

$$\mathbf{R}'(s, s') = \begin{cases} \mathbf{R}(s, s') & \text{if} \quad s \models A \wedge s' \models (A \vee B) \\ 0 & \text{otherwise} \end{cases} \tag{3.26}$$

$$\mathbf{R}'(s, s_{\text{DEADLOCK}}) = \begin{cases} \sum_{s' \not\models (A \vee B)} \mathbf{R}(s, s') & \text{if } s \in S \setminus T \\ 0 & \text{if } s \in T \end{cases} \tag{3.27}$$

Then the probability of satisfying $A \ \mathtt{U}^{[0,t]} \ B$ is equal to the probability of reaching the set of target states T by time t in the modified CTMC $\mathcal{C}'$ for $S' = S \cup \{s_{\text{DEADLOCK}}\}$, that is:

$$\mathtt{P}_{=?}[A \ \mathtt{U}^{[0,t]} \ B] = \sum_{s \in T} \pi_t(s) \tag{3.28}$$

where π_t is the state probability distribution at time t in $\mathcal{C}'$.

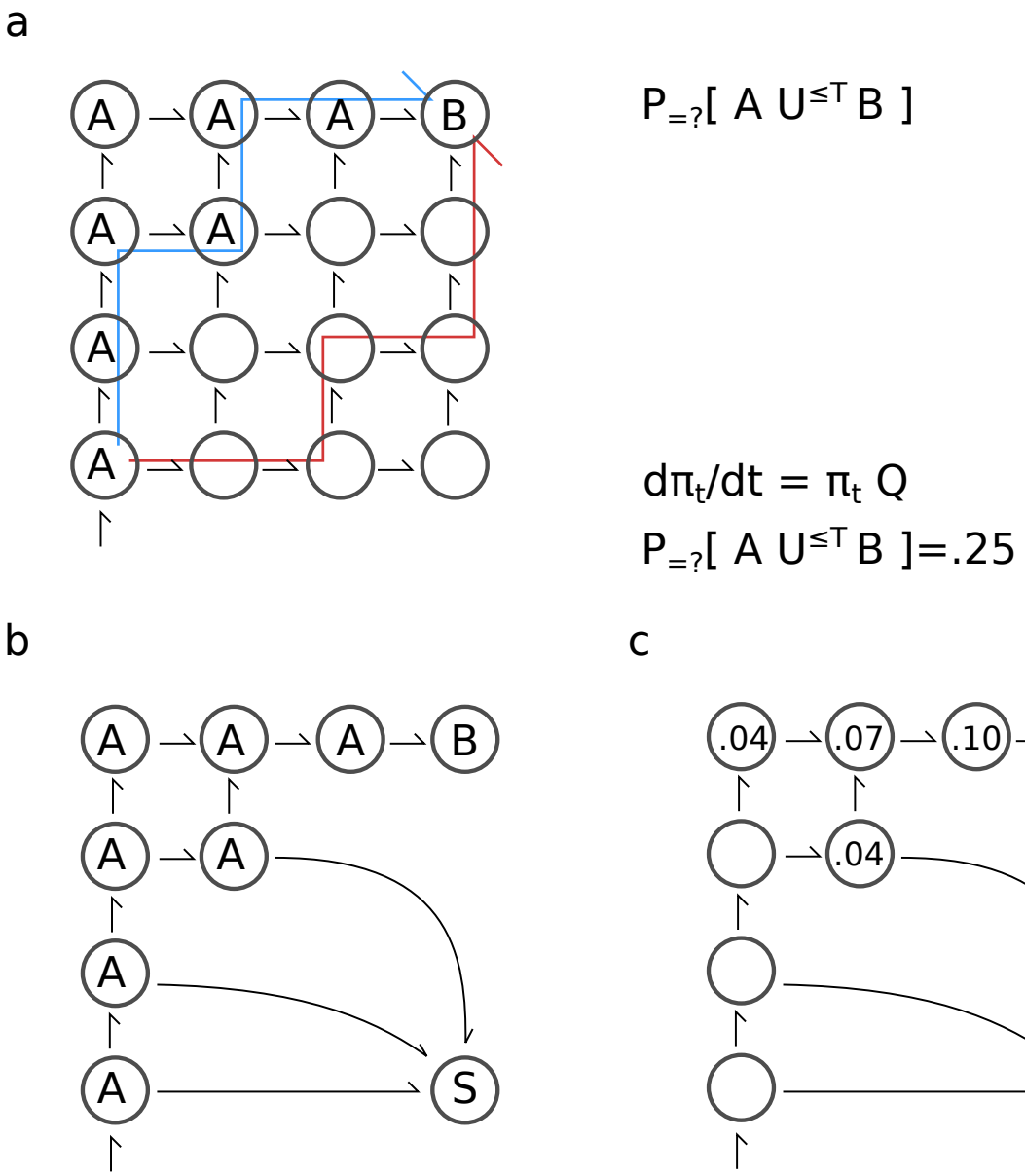


Figure 3.4: Graphical representation of time-bounded probabilistic model checking. a) States in the CTMC are labelled with atomic propositions A and B. Two paths, each starting in the initial state s_0 (bottom left) eventually reach the state labeled with 'B' within T time units. Paths over the Markov chain that do not leave states labelled with 'A' until they reach a state labelled with 'B' within T time units, satisfy the CSL path property $A\ \mathtt{U}^{\leq T} B$. In this case, the blue path satisfies the property, but the red path does not. b) To compute the probability of a path satisfying $A\ \mathtt{U}^{\leq T} B$, the model checking algorithm redirects all edges from states satisfying A into states that do not satisfy A into the sinkstate 'S'. All states not labelled A and B are then removed from the model, with the exception of the sink state. c) Integration the probability density over the ODE specified by the modified CTMC up until time T, given the initial condition of $\pi_{t=0}(s_0) = 1.0$, shows that 25% of the paths satisfy the path-property, so that $\mathtt{P}_{=?}[A\ \mathtt{U}^{\leq \mathtt{T}} B] = 0.25$.

Name	Shorthand	CSL formula
Logical or	$\phi_1 \vee \phi_2$	$\neg(\neg\phi_1 \wedge \neg\phi_2)$
Future	$\mathtt{F}^{\leq T}\phi$	$\mathtt{true}\ \mathtt{U}^{[0,T]}\ \phi$
Instantaneous	$\mathtt{F}^{T}\phi$	$\mathtt{true}\ \mathtt{U}^{[T,T]}\ \phi$
Until	$\phi_1\ \mathtt{U}^{\leq T}\phi_2$	$\phi_1\ \mathtt{U}^{[0,T]}\ \phi_2$

Table 3.2: Shorthand CSL notation.

Nested formulas, for example when A or B in Ex. 3.2.3 is of the form $\mathtt{P}_{\sim p}[\phi]$, require an additional analysis step to determine which states satisfy the inner property.

The transient probabilities can be obtained in one of two ways: either via numerical computation, which we call exhaustive model checking, or stochastic simulation, which we call statistical model checking. Exhaustive model checking of time-bounded CSL properties relies on uniformisation and using this method precise answers can be computed: for example, we might compute the probability of deadlock for a DNA walker to a precision that vastly exceeds the uncertainty in the model parameters. For statistical model checking, which relies on generating a large number of simulations and checking each against the property, the model size is not a limiting factor, but precision is decreased. In particular the probability of rare events is hard to estimate reliably based on stochastic simulation.

Property specification for CTMCs, including the use of sink-states, was a known technique well before the formulation of CSL [98]. We emphasize the benefits of an automated procedure of probabilistic model checking, that is, transitions are not manually set each time a property is analysed but rather the model is stored in a software tool (such as PRISM) that enables CSL model checking. Automated model checking reduces the chance of operator error and enables the integration of model checking in other automated systems. It also enables model checking of probabilistic systems that would be too laborious to perform by hand. PRISM [106] supports both exhaustive model checking (uniformisation) and statistical model checking for CSL.

A CTMC can be augmented with a reward structure, which is an additional annotation to its states and transitions. Formally, a reward structure for a CTMC is a pair $(\rho, \boldsymbol{\iota})$ given by

- $\rho : S \to \mathbb{R}_{\geq 0}$ is the *state reward function*;
- $\boldsymbol{\iota} : S \times S \to \mathbb{R}_{\geq 0}$ is the *transition reward function.*

We now extend the CSL definition with reward properties, and give two examples. PRISM [106] supports CSL reward operators.

Definition 3.2.4 (Continuous Stochastic Logic with rewards)**.** *For CTMCs extended with a reward structure* $(\rho, \boldsymbol{\iota})$ *the cumulative reward and instantaneous reward operators [107, 96] are*

given as:

$$\mathtt{R}_{\sim r}[\mathtt{C}^{\leq t}] \mid \mathtt{R}_{\sim r}[\mathtt{I}^{=t}] \tag{3.29}$$

where $r \in \mathbb{R}_{\geq 0}$ *is a rewards threshold,* $t \in \mathbb{R}_{\geq 0}$ *is a time bound and* $\sim \in \{<, \leq, \geq, >\}$*. Give a state* $s \in S$ *the satisfaction relation is defined by:*

$$s \models \mathtt{R}_{\sim r}[\mathtt{I}^{=t}] \Leftrightarrow E_s[X_{\mathtt{I}^{=t}}] \sim r \tag{3.30}$$

$$s \models \mathtt{R}_{\sim r}[\mathtt{C}^{\leq t}] \Leftrightarrow E_s[X_{\mathtt{C}^{\leq t}}] \sim r \tag{3.31}$$

where the instantaneous and cumulative rewards are given by the random variables $X_{\mathtt{I}^{=t}}$ *and* $X_{\mathtt{C}^{\leq t}}$*, and their expectation* E *are computed as follows*

$$E_s[X_{\mathtt{I}^{=t}}] = E[\rho(\omega@t)) \,|\, \omega \in \ Path(s)] \tag{3.32}$$

$$E_s[X_{\mathtt{C}^{\leq t}}] = E\left[\sum_{j=0}^{j_t-1} (t_j \rho(s_j) + \iota(s_j, s_{j+1})) + \left(t - \sum_{j=0}^{j_t-1} t_j\right) \cdot \rho(s_{j_t}) \middle| \omega \in \ Path(s)\right] \tag{3.33}$$

for $j_t = \min_j \sum_{i=0}^{j} t_i \geq t$*. Additionally,* $\sim r$ *can be replaced by* $=$*? (as opposed to the Boolean valuation of the reward value meeting* $\sim r$*). A state distribution* $\pi : S \to [0, 1]$ *is defined to satisfy the reward operator* $\mathtt{R}_{\sim r}$ *by fair weighting of each state:*

$$\pi \models \mathtt{R}_{\sim r}[\phi] \Leftrightarrow \left(\sum_{s \in S} \pi(s) E_s[\phi]\right) \sim r \tag{3.34}$$

for $\phi \in \{\mathtt{I}^{=t}, \mathtt{C}^{\leq t}\}$*.*

In Fig 3.5 a simple CTMC model is annotated with transition rewards. We now give two more examples of reward properties.

Example 3.2.5 (Cumulative reward property)**.** *Consider the expected cumulative reward by time* t

$$\mathtt{R}_{=?}[\mathtt{C}^{\leq t}] \tag{3.35}$$

for any CTMC $\mathcal{C}$ *with the set of atomic propositions* $AP = \{\text{actionable}\}$ *and the state reward function* ρ

$$\rho(s) = \begin{cases} 1 & \textit{if} \quad \text{actionable} \in L(s) \\ 0 & \textit{if} \quad \text{actionable} \notin L(s) \end{cases} \tag{3.36}$$

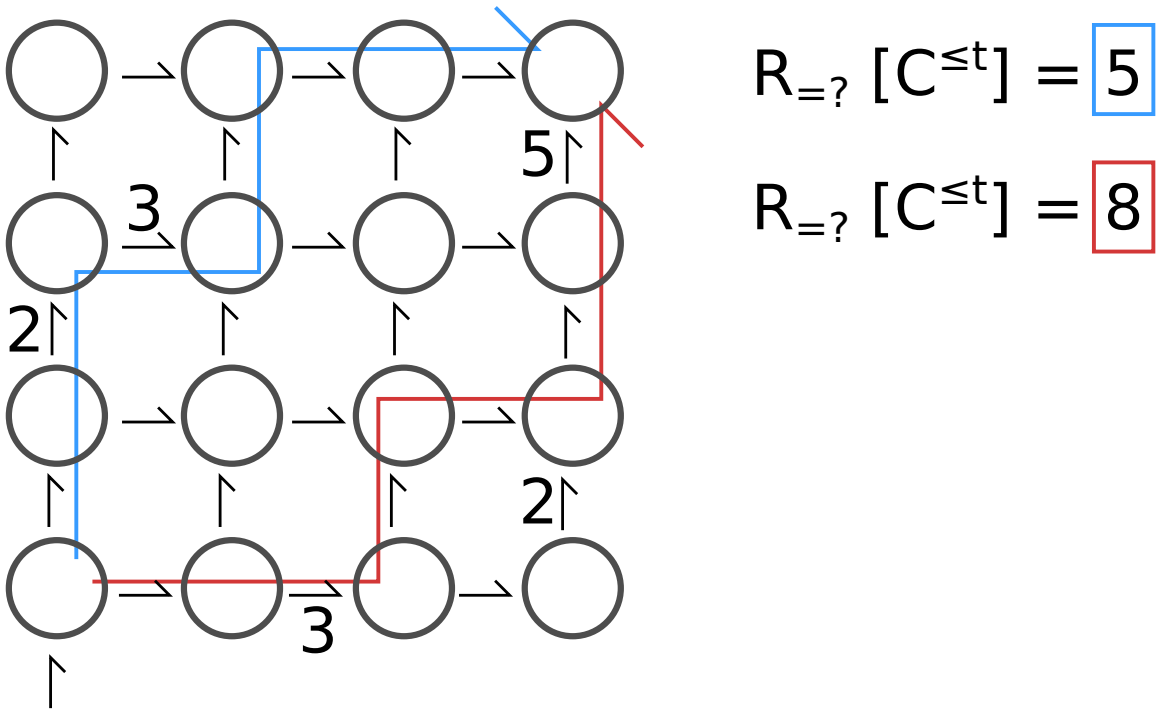


Figure 3.5: Graphical representation of a reward property $\mathtt{R}_{=?}[\mathtt{C}^{\leq \mathtt{T}}]$. In this CTMC, some transitions are labelled with transition rewards. The rewards accumulated by the blue path sum to 5, the rewards accumulated by the red path sum to 8. The CSL reward property $\mathtt{R}_{=?}[\mathtt{C}^{\leq \mathtt{T}}]$ is equal to the expected reward accumulated within T time units for all paths starting in the initial state.

and transition reward $\boldsymbol{\iota}$ zero for all transitions. Then $\mathtt{R}_{=?}[\mathtt{C}^{\leq t}]$ is equal to the expected time spent in actionable states until time t. Through uniformisation the value is given by [108]:

$$\mathtt{R}_{=?}[\mathtt{C}^{\leq t}] = \sum_{s \in S} \frac{1}{q} \sum_{i=0}^{\infty} \rho(s)\tau^i(s) \sum_{n=i+1}^{\infty} \gamma_{n,qt} \tag{3.37}$$

where q is the uniformisation constant and $\tau_i, \gamma_{n,qt}$ the discrete transient vector and Poisson probabilities from Def. 3.1.10.

Example 3.2.6 (Instantaneous reward property)**.** *Consider the expected instantaneous reward at time t*

$$\mathtt{R}_{=?}[\mathtt{I}^{=t}] \tag{3.38}$$

for the CTMC and reward structure in Ex. 3.2.5. Then $\mathtt{R}_{=?}[\mathtt{I}^{=t}]$ is equal to the probability to be in an actionable state at time t. The value is given by

$$\mathtt{R}_{=?}[\mathtt{I}^{=t}] = \sum_{s \in S} \rho(s)\pi_t(s) \tag{3.39}$$

where π_t is the vector of state probabilities at time t.

3.3 Literature review

We summarise relevant related work in probabilistic model checking.

3.4 Probabilistic and statistical model checking

Formal verification

With the development of computer programming came the desire to verify software to some degree of correctness. For example, a program is 'safe' if no execution of the program causes failure, and the program displays 'liveliness' (now known as liveness) if the program eventually returns the desired output [109]. Such properties are especially difficult to prove for concurrent programs, where separate threads operate on memory in an unspecified order. Early on, the Floyd-Hoare logic was developed to reason about correctness of programs. A manual or mechanically constructed proof of correctness based on that, or derived logics, is a typical form of formal verification, which encompasses any method to construct proof of correct operation of software or otherwise. Related are the temporal logics used to express correctness (e.g., safety or liveness) which come in two variants, linear time (e.g., Linear Time Logic [110]) and branching time (e.g., Computational Tree Logic [111]). Formal verification has found application in industry, and we

give two examples. Intel formally verifies parts of processor microcode and other architecture prior to prototype production, while Microsoft validates driver programs that interface between devices and the operating system [112, 113].

Model checking was proposed as an automated alternative to proof construction, and exhaustively verifies specifications expressed in temporal logics over the program's state space, an approach pioneered in [114]. Work by Clarke et al. shows how concurrent systems are model checked [115], albeit significant issues with scalability remain. A state space explosion occurs when the system contains many components or variables. Methods that operate on each state separately become prohibitive when employed, without further refinement, to complex systems. Clarke et al. then proposed the use of symbolic representation to combat the state-space explosion [116]. This relies on seminal work by Bryant, who developed the modern notion of Binary Decision Diagrams (BDD) [117], a data structure that represents Boolean functions efficiently and facilitates the encoding of sets of states as Boolean functions.

An important concept in formal verification is that of counter-examples: to fix the program, the method should return a description of how the specification is violated, in other words proving the violation. Related is the method of counter-example guided abstraction refinement (CEGAR) [118, 119]. In this approach, an attempt is made to verify correctness for an abstract representation of the program by successive refinements until a counter-example is found.

Probabilistic model checking

Probabilistic model checking aims to answer if a specification, expressed in temporal logic, holds for a probabilistic program. Early attempts at the verification of probabilistic programs focussed on verifying that a temporal property holds almost surely (with probability one) over a discrete-time Markov chain [120, 121]. When a process operates in continuous time, a continuous-time Markov chain (CTMC) is well suited to modelling it, and we here focus on the development of probabilistic model checking for continuous-time Markov chains.

The branching time logic CSL was proposed by Aziz et al. [105] to express time-bounded properties over CTMCs. The logic was subsequently extended to describe steady-state behaviour [122] and expected rewards [107]. The contribution of Aziz et al. was to show that time-bounded CSL properties are decidable. It was later demonstrated that the evaluation of time-bounded CSL reachability reduces to transient analysis, that is, computation of the probability to be in each state of the Markov chain at some point in time, starting from initial conditions [97].

The derivation of transient probabilities for continuous-time Markov chains was already studied for the benefit of operations research and performance modelling, where the application

of Runge-Kutta (numerical integration) was compared to uniformisation [98, 108]. Because the computation of transient probability is equivalent to solving a matrix exponential, many methods apply, and for example Krylov subspace methods were also considered [123]. Uniformisation, a method developed by Grassman [99], is the preferred method for its easily derived error bounds and relatively stable performance on non-stiff models [108], while stiff models may still benefit from implicit integration schemes [98].

To combat the problem of state-space explosion various symbolic and explicit approaches have been suggested. The sliding window method is a non-symbolic method developed by Henzinger et al., which works by only keeping a 'window', a subset of states, in main memory. The main idea is that the vast majority of states are unlikely to be visited by the Markov process, and a good approximation results when these states are dropped from the model. As the time evolves, the probability of visiting certain sets of states may increase and decrease for other sets of states, and the use of a sliding window is proposed to accommodate this. The fast adaptive uniformisation method (FAU, [124]) is similar but discards/loads states individually, instead of groups of states, and is based on adaptive uniformisation, a generalisation of uniformisation that allows an adaptive time step [125]. The symbolic approach (BDD-based) was applied to probabilistic model checking and works especially well when the state space is regular to some degree [122]. Using this method, CTMCs with over 33 million states were model checked [126]. Hybrid methods have been proposed since, combining explicit state and BDD representation [127].

A second method of probabilistic model checking is statistical probabilistic model checking, which generates random traces from the Markov chain, and bypasses the state-space explosion problem. Here the transient probabilities are estimated using Monte Carlo methods, as opposed to uniformisation. Traces are simulated using the stochastic simulation algorithm described by Gillespie [102] (also Def. 3.1.12), and each random trace is checked whether it meets the specification. Afterwards, the estimated satisfaction probability and a confidence interval are constructed as is usual in statistics ([128, 129], also see preliminaries in [130]). For example, take a CSL property in the form of $\mathtt{P}_{\leq c}[\phi]$. After simulating a number of traces, the probability of satisfying the CSL path formula ϕ is estimated to lie within a region (a, b) given a preset confidence level α. PRISM also supports methods that continue to simulate traces until the satisfiability of a CSL property can be decided, in other words, in this mode PRISM will continue to simulate until the bounds (a, b) are narrow enough to exclude c. This mode is based on Wald's sequential probability ratio test [128].

Probabilistic model checking software

The PRISM probabilistic model checking tool [106] supports automatic verification of discrete-time Markov chains and continuous-time Markov chains in addition to three other model types, and accepts models created in the PRISM modelling language as well as the Systems Biology Markup Language (SBML). Symbolic or explicit data structures can be used to verify CSL or PCTL properties. PRISM was used to verify part of the Bluetooth and Firewire communication protocols, which contain probabilistic elements [131, 132], in addition to case studies of cellular signalling pathways (FGF and MAPK) [133, 134].

PRISM supports four analysis engines in addition to the simulation mode [135]: the purpose of all these engines is to check satisfiability of CSL formulas over Markov chains. The analysis engines all employ standard uniformisation and differ only in the internal representation of the state-space. Two of the engines are based on symbolic methods, and rely on Binary Decision Diagrams (BDDs) or Multi Terminal Binary Decision Diagrams (MTBDDs), and we list all four here:

- The `sparse` engine relies on symbolic representation and uses sparse matrices.
- The `MTBDD` engine wholly relies on MTBDD and BDD representation.
- The `hybrid` engine combines elements from the `sparse` and `MTBDD` engines.
- The `explicit` engine represents states and rates in an explicit (non-symbolic) fashion.

Owing to the non-symbolic representation, the `explicit` engine is easier to extend and modify, and this is the engine that we use to implement the methods described in Chapter 5. Symbolic engines perform well when the state space has a repetitive or regular structure.

Other examples of probabilistic model checking software are MRMC [136], MARCIE [137], the Bio-PEPA project [138], COSMOS [139] and SHAVE [140]: this list is non-exhaustive. Different tools have different aims: COSMOS accept models in the form of Petri nets and specifications in the form of (generalized) Deterministic Timed Automata (DTA), where we note that PRISM also accepts DTAs. MARCIE is a tool for the analysis of Petri nets. SHAVE is a tool for chemical reaction networks that combines discrete stochastic and continuous stochastic representations.

In this thesis, all development and prototyping related to probabilistic model checking was done in PRISM. Because PRISM is developed at the University of Birmingham and the University of Oxford, excellent developer support was available.

Chapter 4

Computation with DNA walkers

Logic gates based on DNA strand displacement (see Section 2.3) were recently created and used to implement small logic circuits, such as an artificial neural network, a 4-bit square root computer and an approximate majority consensus algorithm [11, 12, 16, 17]. In this approach, each gate is a complex of DNA strands and the gates communicate via, and operate on, signal strands, which results in a massively parallel execution: billions of copies of each gate consume and produce similar amounts of signal strands during operation. To create larger circuits, multiple gates are combined in a single solution. An improvement over this approach is proposed in the form of localized environments that each contain a single copy of each gate and where each localized system executes in isolation from others. It might be possible, for example, to prepare origami tiles with a single copy of each gate tethered to it, so that the gates are in close vicinity of one another [11, 141, 142]. The advantage of the local approach is twofold:

- Diffusion of signal strands is no longer required when the communicating gates are in the same location. This leads to faster operation.
- The collection of gates operates in isolation, preventing unintended cross-talk that might lead to faulty behaviour. In the case of gates tethered to a origami tile, this can be achieved by fixating the tiles onto a surface.

A drawback to the local approach is the propagation of error: the operation of the entire localized unit is compromised when a single gate fails.

In this light we investigate artificial molecular walkers, made from DNA, for their applicability as localized computational device. DNA walkers are man-made molecular motors that mimic the functionality of proteins that traverse microtubules within the cell, such as kinesin and dynein. Many different systems have been proposed [46, 43, 47, 48, 49, 50] but we study the DNA walker developed by Wickham et al. [44, 45], which is attractive for its innovative method to direct the walker along junctions in the track. We develop a stochastic model based

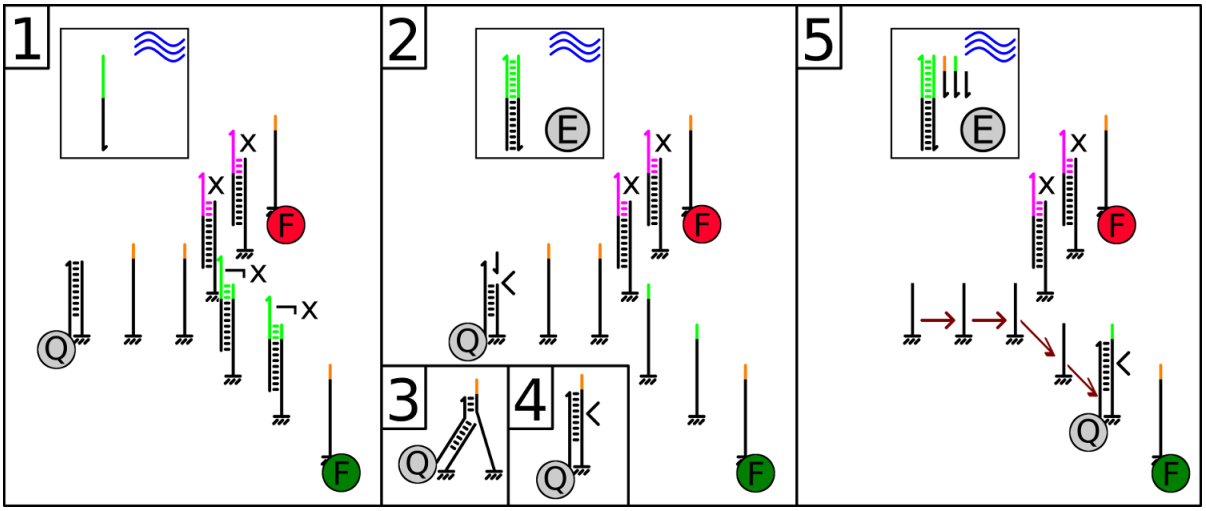


Figure 4.1: From [4]. (1) The walker strand carries a quencher (Q) that will quench fluorophores (F) when nearby. After experimental preparation, the walker is attached to the initial anchorage and blocking strands are present on designated blocking anchorages. Selected anchorages that are initially blocked can become unblocked by adding complementary unblocking strands. In this case, unblocking strands are added for the blocked anchorages that are labelled by $\neg X$. (2) Once a nicking enzyme (E) is added, it can attach to the walker-anchorage complex and cut the anchorage. The anchorage top melts away from the walker, exposing 6 nucleotides as a toehold. (3) The exposed toehold becomes attached to the next anchorage. (4) In a displacement reaction, the walker migrates to the new anchorage. The stepping is energetically favourable, because it re-forms the base pairs that were lost after the previous anchorage was cut. (5) Repeating this process, the walker arrives at a junction. The walker continues down the unblocked track, eventually reaching the final anchorage and quenching the fluorophore

on existing experimental work and demonstrate the merit of probabilistic model checking to analyse their reliability, performance and correctness, and propose design principles for walker circuits that reduce leakage. The developed model and the associated PRISM files appear in a case study for the PRISM project [1], highlighting both the DNA walker model and the merits of the fast adaptive uniformisation method that is covered in Section 5.1.

The chapter is organised as follows. In Section 4.1 we describe the walker and discuss its computational applicability. A probabilistic model of the walker is developed in Section 4.2. In Section 4.3 we investigate the performance of the walker and identify leakage rates that can lead to failure. In Section 4.4 we state design principles for DNA walker circuits that optimise its behaviour. We summarize the chapter in Section 4.5.

This chapter is derived from published work [3, 4].

4.1 A man-made molecular motor

In this design the walker consists of a single strand of DNA that traverses a series of strands (anchorages) tethered to a DNA origami tile, and as a result the relative positioning of anchorages

is restricted to triangular lattice. The walker is complementary to the anchorages and steps from one anchorage to the next via toehold mediated strand displacement (also see Fig. 4.1). When the walker steps onto a new anchorage, a nicking enzyme eventually attaches to the walker-anchorage duplex and nicks the anchorage, re-exposing the toehold domain on the walker. This mechanism prevents the walker from moving onto previously visited anchorages: nicked anchorages no longer contain the toehold domain that is required to initiate stepping. Some anchorages include deliberate nucleotide mismatches, which prevents the enzyme from nicking. Consequently the walker does not step away from these anchorages, and we refer to these as absorbing or final anchorages.

After experimental preparation, all anchorages are unblocked, that is, they are hybridized to the origami and no other strand, with the exception of designated blocking anchorages that are initially occupied by a blocking strand. Blocking strands are addressed by means of distinct toehold sequences: anchorages are selectively unblocked by adding strands complementary to blocking strands (also Fig. 4.1). By using programmable anchorages at track junctions, Wickham et al. [45] demonstrate that a walker can be directed to any leaf in a complete two-level binary tree using input strands that unblock the intended path.

Computational circuits

We now discuss a method to encode Boolean formulas as walker circuits. In this approach, a circuit is initially prepared and the computation starts when input and the nicking enzyme are added. At the end of the computation, the circuit can return a `true` or `false` signal. Computations that do not result in any signal are said to be incomplete. Executions that never return a signal, no matter how long we wait, are said to be deadlocked.

A track is a set of anchorages along a line, as in Fig. 4.4 (see 'full track'), and a circuit is a set of tracks that contain initially blocked junctions. Each prepared circuit contains a single walker-anchorage complex: this anchorage is the initial position of the walker. In this chapter we discuss, in addition to others, a circuit with double junctions (Fig. 4.2). The last anchorage of a track that does not lead onto a junction is called a final anchorage, and is made absorbing: once the walker steps onto an absorbing anchorage, it can no longer move. Each absorbing anchorage is given a truth assignment, so that the output of the circuit is equal to the truth assignment of the final position of the walker. If the walker does not reach an absorbing anchorage by the end of the experiment, then the output for that circuit is undefined. If the walker is in a position where it can no longer move, but has not reached an absorbing anchorage, then we call the circuit deadlocked. Our model only allows three-way junctions: these were demonstrated to work in [44, 45].

Reporting strategy

In the experimental demonstration, the walker carries a quencher (IabRQ) that quenches the response of fluorophores attached to the absorbing anchorages (see Fig. 4.1). The distance at which the energy transfer between the fluorophore and quencher is at 50% efficiency, R_0, are in one study [143] measured as $R_0 = 3.0 \pm 0.6$ nm and $R_0 = 3.0 \pm 0.7$ for the fluorophore-quencher pairs Cy3-IabRQ and TAMRA-IabRQ. Given that average distance between anchorages is approximately 6.2 nm, we assume that significant quenching is only observed when the walker is attached to a final anchorage, regardless of which fluorophore is used.

Our intended computational application, the embedding of Boolean formula as walker circuits (Thm. 4.1.7), results in many final anchorages to appear within a single circuit. A problem arises when we perform computation with a circuit that contains many final anchorages, because the walker can quench only one of them. As a result, the signal-to-noise ration is worse for circuits with many final anchorages. To avoid this problem, we propose an alternative reporting strategy.

One approach is to attach a flourophore to the walker, and to attach quenchers to each absorbing anchorage that evaluates to `true`, as in Fig. 4.2. When the computation evaluates to `true`, a decrease in signal is observed, while maintaining a high signal-to-noise ratio. However, when the computation outputs `false`, no reduction in fluorescence is observed, and we thus cannot distinguish between a stalled (deadlocked) execution, and a computation that outputs `false`.

We propose to use a dual-rail reporting strategy, so that a computation evaluating to `true` reduces the signal measured in one circuit, and computations that evaluate to `false` reduces the signal measured in another circuit. If the two circuits are prepared so that the fluorophores, which are attached to the walkers of each circuit, are spectrally distinguishable, then both circuits could be mixed in a single solution without problem. Our approach is illustrated in Fig. 4.2. The two circuits have the same topology and design, and differ only in the location of quenchers and the type of fluorophore attached to the walker.

In the remainder of this work, we assume that additional final anchorages do not prevent a practical implementation of our circuits, but we avoid discussing the two variants of each circuit.

Computational expressiveness

It is not difficult to see that any Boolean formula can be simulated by a walker circuit: the truth-assignment for a formula over n variables can be embedded in the form of a binary tree of depth at most n. In this construction the worst-case number of end-nodes is exponential in the number of variables (2^n). We now discuss an embedding of Boolean formula that requires an amount of end-nodes that scales only linear with the number of variables. This requires the

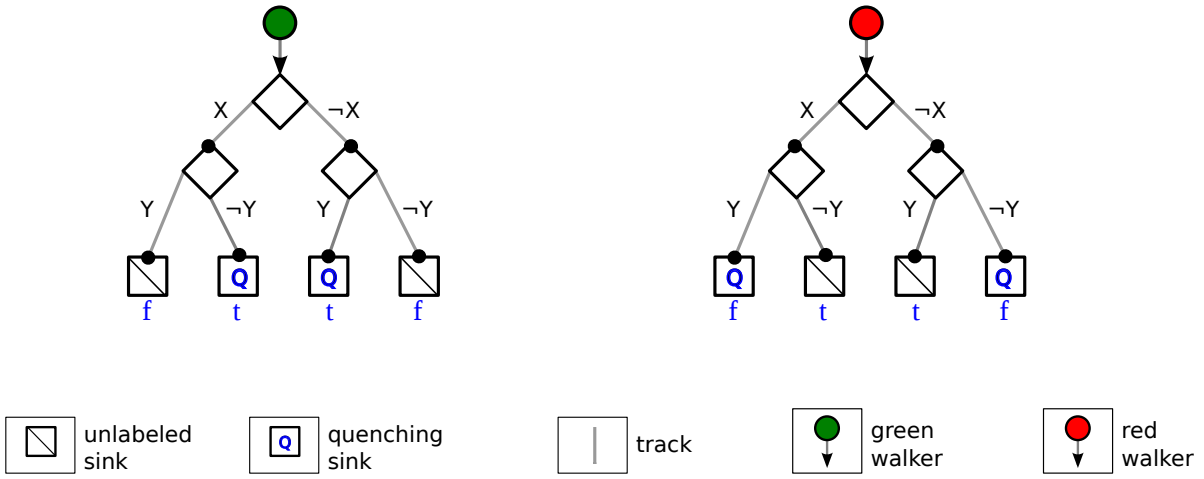


Figure 4.2: Modified from [4]. DNA walker circuits implementing a XOR logic gate. Two sets of circuits are prepared, which differ only in the fluorophore (which is attached to the walker) and the positioning of quenchers on the final anchorages. The two types of fluorophores are spectrally distinguishable, and for convenience they are labelled red and green in this example. In one circuit quenchers are attached to the end-nodes that indicate `true`, while in the other circuit, quenchers are attached to end-nodes that indicate `false`. Adding strands indicating $X =$ `true`, $Y =$ `false` opens the tracks labelled with X and $\neg Y$. After adding the nicking enzyme, the walker moves left on the first junction and right at the second junction in both circuits. The fluorescence response from green fluorophores drops proportionally to the number of walkers that reach the end-node, while the red signal remains constant. If the circuit with the green fluorophore is executed in isolation, then computations evaluating to `false` are indistinguishable from stalled executions.

formula to be in 3-conjunctive normal form (3CNF):

Definition 4.1.1. *A 3CNF formula over m clauses and n variables is a Boolean formula F*

$$F = C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_m \tag{4.1}$$

$$C_i = K_i \vee L_i \vee M_i \tag{4.2}$$

where K_i, L_i, M_i are literals over the variables, e.g., K_i is a variable or the negation of a variable.

The embedding of a 3-conjunctive normal form (3CNF) formula with m clauses requires only $\mathcal{O}(m)$ many end-nodes, which we prove at the end of this section (Thm. 4.1.7). To see that the embedding is possible, observe in Fig. 4.8a an embedding of the following 3CNF formula with $n = 9, m = 3$:

$$(X \vee Y \vee Z) \wedge (\neg R \vee S \vee \neg Y) \wedge (Q \vee \neg V \vee Z). \tag{4.3}$$

The circuit of Fig. 4.8a is composed of three smaller circuits that are connected in series, and each clause corresponds to one of the smaller circuits.

This embedding is attractive for two reasons:

- Any Boolean formula can be converted into 3CNF, and
- The embedding re-uses the same topological design for each clause (Fig 4.3), meaning that the optimization of this single circuit improves the performance of every 3CNF embedding.

Also note that the problem of satisfiability of 3CNF is NP-complete [144], meaning that any NP problem can be embedded as a DNA walker circuit. If some input exists such that the walker reaches a final anchorage that is labelled `true`, then the problem is satisfiable. It is unclear at this stage if randomized search algorithms could be implemented using a molecular computer.

From Section 4.2 onward, we develop a model of walker behaviour and propose design principles (Section 4.4) that increase performance. To demonstrate the second point in the above, we apply the design principles to the smaller circuit, and obtain improved performance on the composed circuit (Fig. 4.8b). The improved performance is demonstrated in Table 4.4. We now demonstrate our main result regarding the computational expressiveness of DNA walker circuits.

Definition 4.1.2 (DNA walker circuit)**.** *A DNA walker circuit is a touple (V, E, M, L, O) of a graph (V, E), set of Boolean variables M, a labeling function from the set of edges to a literals over M while also permitting the empty label ϵ for unblocked tracks, $L : E \to (lit(M) \cup \{\epsilon\})$ and an output function O. A vertex $v \in V$ is either the initial position of the walker $v_0 \in V$, a final*

anchorage $v \in V_F$ or a junction vertex $v \in V_J$, that is,

$$v_0 \notin V_F \tag{4.4}$$

$$v_0 \notin V_J \tag{4.5}$$

$$V_A \cap V_J = \emptyset \tag{4.6}$$

$$V = \{v_0\} \cup V_F \cup V_J \tag{4.7}$$

Further, the initial vertex and final vertices have one edge, $|v_0| = 1$ and $\forall v \in v_F : |v| = 1$, and junction vertices have three edges, $\forall v \in V_J : |v| = 3$. The output function $O : V_F \to \{T, F\}$ provides a truth assignment for each of the final vertices.

In order to encode Boolean functions as walker circuits, we require that the output of the DNA walker circuit is deterministic:

Definition 4.1.3 (Span of a DNA walker circuit under a variable assignment)**.** *Given a DNA walker circuit (V, E, M, L, O) and truth assignment of the input variables $f : M \to \{T, F\}$, the span is given by the implied graph (V, E') of unblocked edges, where*

$$e \in E' \iff f(L(e)) = T \qquad \text{where}$$

$$f(\epsilon) = T \tag{4.8}$$

Definition 4.1.4 (Deterministic walker circuit)**.** *A DNA walker circuit (V, E, M, L, O) is deterministic if, under any truth assignment of the input variables, only one path between the initial vertex and any of the final vertices exists. The output of the circuit is equal to the labeling of the reachable final vertex under the output function O. That is, given a truth assignment over the input variables M and resulting span (V, E'), the initial vertex v_0 and exactly one $v \in V_F$ are connected, and any junction $v \in V_J$ reachable from the initial vertex v_0 has exactly two edges.*

We also assume that walker circuits are always planar: although feasible, crossed tracks were not part of the experimental demonstration [44, 45], and it is not obvious how the performance of the walker would be impacted, if implemented. We also require circuits to be composable, so that any two circuits can be put in series by linking the output of one circuit to the input of a second circuit.

Definition 4.1.5 (Composable walker circuit)**.** *A DNA walker circuit (V, E, M, L, O) is composable if there exists a planar embedding of the graph (V, E) such that the initial vertex v_0 and final vertices $v \in V_F$ are part of the exterior face.*

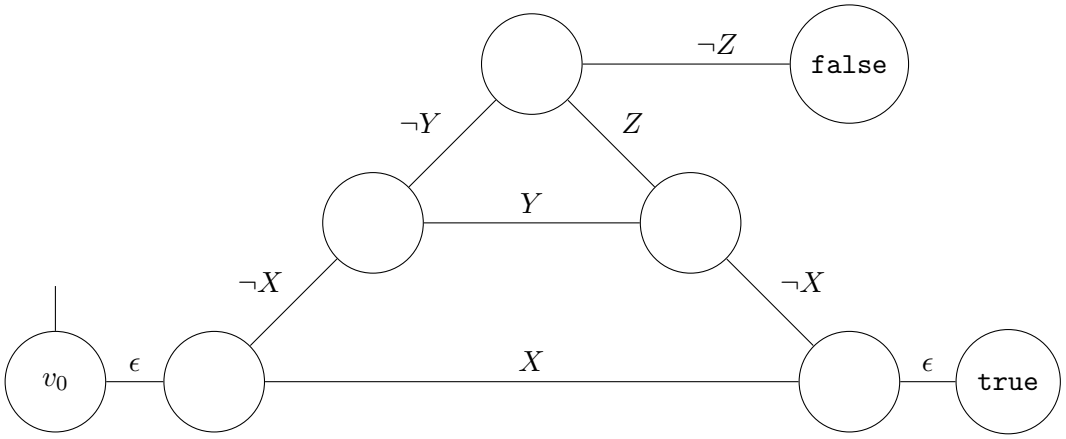


Figure 4.3: A deterministic and composable DNA walker circuit with three junction vertices and two final vertices. This circuit encodes the formula $X \vee Y \vee Z$.

Lemma 4.1.6 (Composition of circuits)**.** *Let $\mathcal{A}$ and $\mathcal{B}$ be deterministic and composable circuits with at least one junction. Let v_A be a junction that shares the edge e_A with a final anchorage of $\mathcal{A}$, and let v_B be the junction that shares the edge e_B with the initial node of $\mathcal{B}$. Then the circuit obtained by adding an unblocked edge between v_A and v_B and removing e_A and e_B is deterministic and composable.*

Figure 4.3 shows a deterministic and composable walker circuit. We now prove the main result of this section.

Theorem 4.1.7. *Given a 3CNF formula with m clauses, there exists a deterministic and composable walker circuit with $\mathcal{O}(m)$ final anchorages, where at most one final anchorage outputs* `true`*, so that the output of the circuit corresponds to the valuation of the formula.*

Proof by induction. (m=1). The 3CNF formula is of the form $X \vee Y \vee Z$ where X, Y, Z are literals over the set of variables. There exists a deterministic and composable walker circuit with one final achorage that outputs `true`, and implements $X \vee Y \vee Z$, see Fig. 4.3. We now assume the theorem holds for m, and demonstrate it holds for $m + 1$.

Without loss of generality, the formula is of the form $\mathcal{F} \wedge (X \vee Y \vee Z)$ where $\mathcal{F}$ is an 3CNF formula consisting m clauses and X, Y, Z are literals. By the induction hypothesis there exists a deterministic and composable walker circuit $\mathcal{C}$ with at most one final anchorage that outputs `true`, that implements $\mathcal{F}$ and has $\mathcal{O}(m)$ many final vertices. If the circuit does not have an final vertex that maps to `true` under output function O, then the formula is trivial. Now assume the circuit contains a final anchorage that outputs `true`.

By Fig. 4.3 there exists a deterministic and composable walker circuit $\mathcal{D}$ with one final anchorage that outputs `true`, and implements $(X \vee Y \vee Z)$. Let $\mathcal{E}$ be the circuit composed of $\mathcal{C}$ and $\mathcal{D}$, obtained by linking the `true` output of $\mathcal{C}$ with the input of $\mathcal{D}$ using the construction

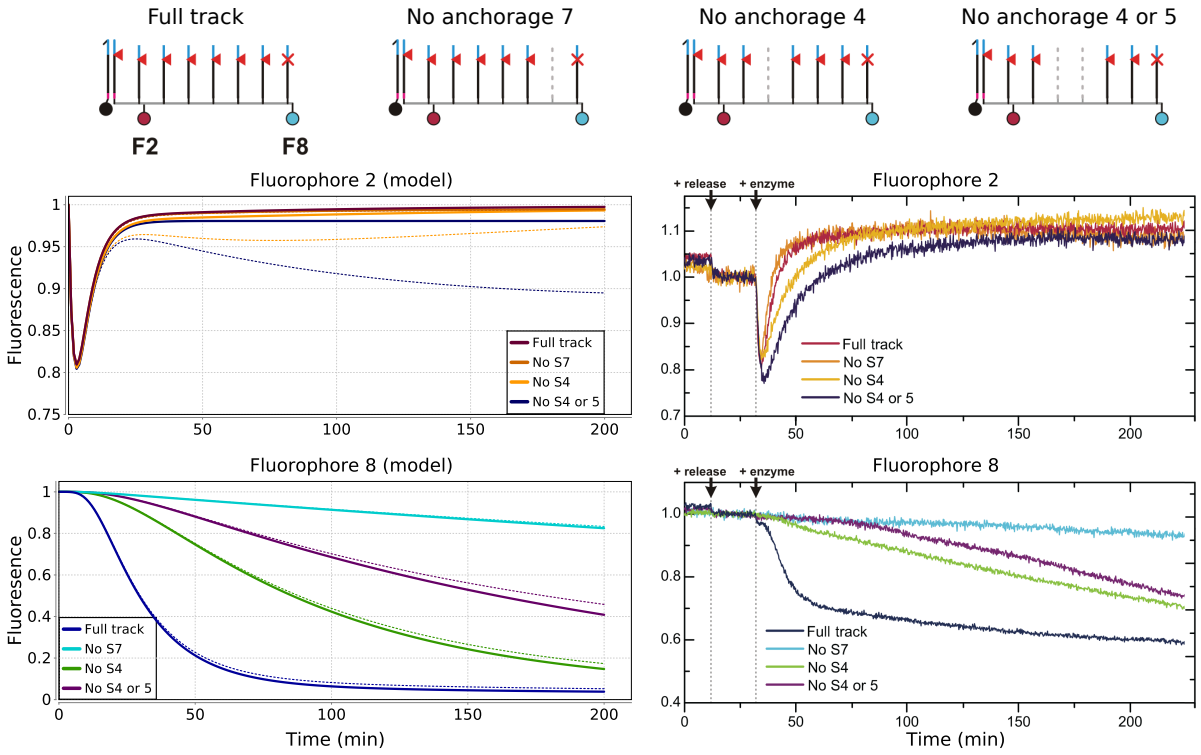


Figure 4.4: Adapted from [3, 4]. Top: A small track of 8 anchorages with fluorophores on both the second and last anchorage. Right: Normalized fluorophore response (reproduced with permission from the authors [44]). The walker hardly reaches the final anchorage when anchorage 7 is removed, due to the double penalty of a longer final step and the mismatch in the final anchorage. Left: The fluorophore luminosity in the model, evaluated at $T = 0, 1, 2, ..$ using standard uniformisation and default settings for PRISM's hybrid engine. Dotted lines: Alternative model where the walker can step onto already-cut anchorages with a reduced rate.

of Lemma 4.1.6, then $\mathcal{E}$ is deterministic and composable. Then $\mathcal{E}$ has one output evaluating to `true`, encodes $\mathcal{F} \wedge (X \vee Y \vee Z)$, and has $\mathcal{O}(m)$ number of final vertices. □

4.2 Probabilistic model of walker behaviour

We now take into account various modes of error that affect the walker and develop a CTMC model of the stepping behaviour of the walker. The model is based on previous DNA walker experiments and a known estimate for the stepping speed of the walker [43, 44, 45]. The predictions of the model match control experiments from [44] qualitatively (Fig. 4.4). To model branched circuits we additionally allow a failure rate for blockades, which we also describe in this section. The blockade failure rate is fitted to measurements on the single junction circuit (see Fig. 4.5a), and to test the quality of the model, we compare the model predictions for the double-junction circuit (see Fig. 4.5c) with the experimental measurements.

In the model, each transition corresponds to the walker changing its location from one

anchorage to the next, skipping over any intermediate steps (Fig. 4.1). We assume non-zero rates for the walker to step onto any intact anchorage within 24 nm distance. This range was chosen by considering the length of an unoccupied anchorage and the length of a walker-anchorage complex, estimated around 15 nm and 11 nm, respectively. The experiment in Fig. 4.4 demonstrates that the walker is capable of stepping across 19 nm gaps, suggesting the maximum interaction range should be at least this distance.

Assume the stepping rate k depends on distance d between anchorages and some base stepping rate k_s. Denote by $d_a = 6.2$ nm the average distance between anchorages in the experiment shown in Fig. 4.4 and let $d_M = 24$ nm be the maximal interaction distance. Based on estimates in the experimental work [44, 45], we approximate the stepping rate k as

$$k = \begin{cases} k_s = 0.009 \text{ s}^{-1} & \text{when } d \leq 1.5d_a \\ k_s/50 & \text{when } 1.5d_a < d \leq 2.5d_a \\ k_s/100 & \text{when } 2.5d_a < d \leq d_M \\ 0 & \text{otherwise.} \end{cases} \tag{4.9}$$

where k_s is the stepping rate of the walker between regularly spaced anchorages, and $k_s/50$ and $k_s/100$ are the reduced stepping rates when the anchorages are at double or triple the normal distance, respectively (see top of Fig. 4.4). These rates define a sphere of reach around the walker-anchorage complex, allowing the walker to step onto an uncut anchorage when it is nearby. Because our model assumes a flat surface, this effectively acts as an radius of interaction, and in Fig. 4.5b this radius is depicted to scale with walker circuits.

We allow two exceptions to the stepping rate, described below.

- Because the complementary domain between the walker and the initial anchorage is two bases longer than usual, the movement is initially slowed down. The stepping rate is reported to be 3× as slow: this is incorporated in the model.
- Absorbing anchorages include a mismatched base, preventing enzymatic activity, but this modification also makes it harder for the walker to attach. Based on the experimental data of Fig. 4.4, we fit a tenfold reduction for the stepping rate using the measurements of the 'no S7' circuit of Fig. 4.4. In that circuit, the jump from the pre-final to the final anchorage occurs at a rate of $k_s/500$ where we multiplicatively apply the penalty of double distance and the mismatch ($1/50 \times 1/10$).

In addition, we deliberately omit three types of known interactions from the model.

- A rate of $k_s/5000$ is reported for transfer of the walker between tracks on separate origami tiles [44]. We conjecture that binding the tiles to a surface eliminates this transfer alto-

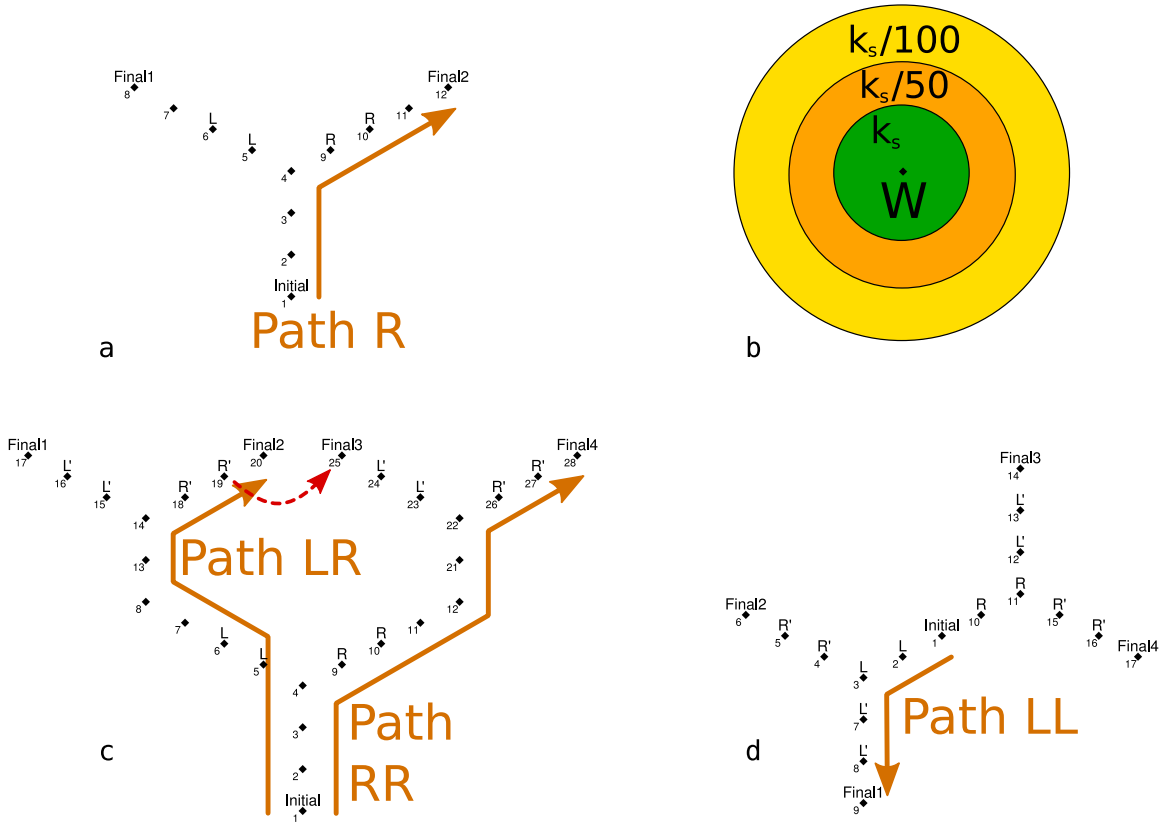


Figure 4.5: Adapted from [3, 4]. Circuit layout for single-junction (a) and double-junction (c,d) decision circuits. Circuit (d) is an improved version of circuit (c). 'Initial' indicates the initial anchorage, 'Final' indicates absorbing anchorages, and letters indicate blocked anchorages. b) Coloured circles indicate the range of interaction of the walker to scale. The red dotted line indicates a leakage transition.

gether.

- The walker can move between intact anchorages, with a rate of approximately $k_s/13$ [44]. We assume the enzyme nicks the intact anchorage quickly, and as a result the walker spends relatively little time attached to non-nicked anchorages. In the model, the anchorage becomes nicked as soon as the walker steps onto it.
- The walker is known to step backward onto cut anchorages. This requires a blunt-end strand-displacement reaction which is known to be slow relative to toehold-mediated displacement [82]. A variant of the model with a backward rate $k_b = k/500$ is shown in dotted lines in Fig. 4.4 (Left). In this case the model predicts significant quenching of fluorophore F2 at late times by walkers whose forward motion is obstructed by omission of one or more anchorages: this does not match experimental data.

The time-dependent responses of fluorescent probes F2 and F8 shown in Fig. 4.4 are predicted by the CTMC model using the above rate parameters without any further fitting: the model qualitatively matches the data. One discrepancy is stands out, in the model the eventual

Property	Formula	Description
Finishes	$\mathtt{P}_{=?}[\mathtt{F}^{T}$ finished]	The probability for the walker to quench any fluorophore.
Finished-correct	$\mathtt{P}_{=?}[\mathtt{F}^{T}$ finished-correct]	The probability for the walker to quench the correct fluorophore.
Finished-incorrect	$\mathtt{P}_{=?}[\mathtt{F}^{T}$ finished-incorrect]	The probability for the walker to quench the incorrect fluorophore.
Correct	$\frac{\mathtt{P}_{=?}[\mathtt{F}^{T}\ \text{finished-correct }]}{\mathtt{P}_{=?}[\mathtt{F}^{T}\ \text{finished}}$	The probability for the walker to return the correct answer.
Deadlock	$\mathtt{P}_{=?}[\mathtt{F}^{T}$ deadlock]	The probability for the walker to get stuck prior to quenching.
Steps	$\mathtt{R}_{=?}[\mathtt{C}^{\leq T}]$	The expected number of steps taken until time T.
Stay-on-path	$\mathtt{P}_{=?}[$correct-path $\mathtt{U}^{\leq T}$finished-correct]	The probability for the walker to stay on the intended tracks and reach the correct final anchorage.
Finish-or-deadlock	$\mathtt{P}_{=?}[\mathtt{F}^{T}(\text{finished} \vee \text{deadlock})]]$	The probability for the walker to either quench a fluorophore or to deadlock.

Table 4.1: CSL specifications for walker behaviour on the single and double-junction circuit. Properties are evaluated for $T = 200$ min. The reward structure for the property Steps assigns a reward of 1 to each transition in the model ($\boldsymbol{\iota}(s, s') = 1\,\forall s, s' \in S$).

	Experiment			Model		
%	R	RR	LR	R	RR	LR
Finishes	65	56	56	97	96	92
Correct	76	87	50	78	85	50
Deadlock				.084	.16	.063
Steps				7.1	7.0	6.6

Table 4.2: Adapted from [3, 4]. Experimental results for the single junction circuit (Fig. 4.5) compared with the model. Properties and CSL formulas are described Table 4.1. Circuit R has a single blocked anchorage on the left, circuit RR has double blocked anchorages on the left, and circuit LR has a single blocked anchorage on both sides. The model significantly overpredicts the probability of the walker to quench any fluorophore ('Finishes'), as expected from the comparison of the fluorophore response in Fig. 4.4. The blockade failure rate, 30%, was inferred by hand by comparing the probability of the walker to quench the correct fluorophore ('Correct'). The blockade failure rate strongly influences the probability of the walker to deviate from the opened up path, but hardly affects the probability of the walker to reach a final anchorage. The model results are obtained using the fast adaptive uniformisation method (Section 5.1 and [5]), resulting in a precision of $< 10^{-6}$.

	Experiment				Model				Optimised model			
%	RR	RL	LR	LL	RR	RL	LR	LL	RR	RL	LR	LL
Finishes	33	40	22	33	90	89	89	90	94	92	94	92
Correct	70	65	55	76	77	74	74	77	78	78	78	78
Deadlock					1.0	1.7	1.7	1.1	0.0	0.0	0.0	0.0
Steps					11.7	11.8	11.8	11.7	5.1	5.1	5.1	5.1

Table 4.3: Adapted from [3, 4]. Experimental results for the double-junction compared with the model. Properties and CSL formula are described Table 4.1. In the 'LR' circuit the anchorages L and R' are unblocked, so that the walker moves left on the first junction, and right on the second, also see Fig. 4.5. The results are generated by checking at least 10^5 simulated paths against each property.

quenching of the final fluorophore (F8) to significantly higher (90 %) than measured (40%) for the full track. Because k_s works as an time-scaling factor for the model, as per Eq. 4.9, fitting this parameter to the data directly will not resolve this problem.

An additional parameter is needed to model branched circuits (Fig. 4.5). We add a possibility of failure of the blocking mechanism, such that, before the walker starts the computation, each blocked anchorage may unblock spontaneously. The probability of unblocking is uniform for all blocked anchorages. The walker can step onto such spontaneously unblocked anchorages, and may thus divert from the intended path. This may delay the walker, or, worse, it may direct the walker to reach a different end-node, leading to the computation returning the wrong result. We infer a failure rate of 30% by fitting to the results ('Correct') of the single-junction experiment, see Table 4.2. The model fails to predict the correct amount of quenching ('Finishes' in Table 4.2), but we refrain from making further changes to the model.

Generating PRISM scripts

A custom software tool was developed to automatically generate PRISM model files, PRISM property files and a circuit graphic based on a scripted description of the circuit. PRISM was then used to verify the specified properties for the generated models. The model files and property files for two walker circuits are included in Appendix A, being:

- The control track without any missing anchorages.
- The single-junction track with two blockades on the left.

The PRISM models, property files and circuit graphics generated for all of the circuits used in the control experiment (Fig. 4.4), and those used to model the single-junction and double-junction circuits (Fig. 4.5) are available for download at [1].

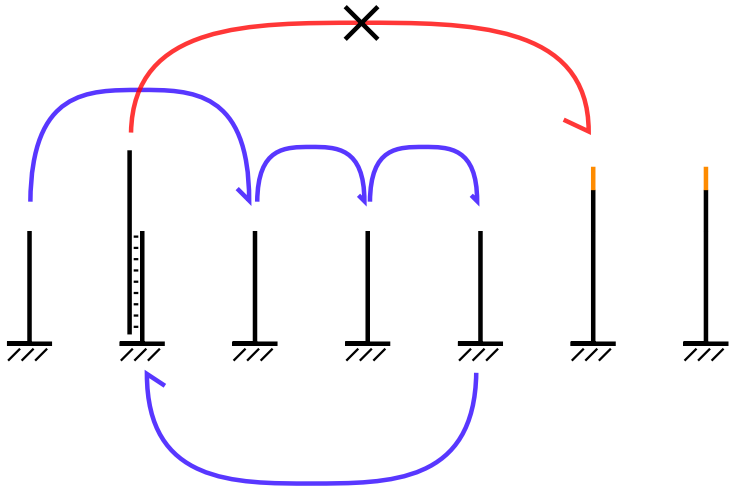


Figure 4.6: From [4]. A proposed mechanism for walker deadlock. After stepping back onto an initially skipped anchorage, there are no intact anchorages left within reach.

4.3 Model results

The straight track and the single junction circuit are used to determine the model and cannot be used to test its predictive quality. We evaluate the model by comparing against data for the double-junction circuit, see Table 4.3. The model captures essential features of the walker behaviour and is reasonably well aligned with the experimental data. In the model, not all walkers reach an absorbing anchorage by time $T = 200$ min, although the predicted quenching is higher than observed, as it was for the single junction circuit.

We investigate walker behaviour by testing against temporal logic queries related to reliability and performance. Not all the walkers that finish quench the intended fluorophore, and in both model and experiment a difference between paths that follow the side of the circuit (paths LL and RR in Fig. 4.5) and paths that enter the interior (paths RL and LR) is observed: the probability of a correct outcome for the side paths is greater. This is explained by leakage transitions between neighbouring paths: an example leakage transition is indicated by a red dotted line in Fig. 4.5(c). Walkers on an interior path can leak to both sides, but a path that follows the exterior can only leak to one side. This effect can also be shown by inspecting paths. The property

$$\mathtt{P}_{=?}[\ \text{correct-path}\ \ \mathtt{U}^{\leq T}\ \text{finished-correct}\ ] \tag{4.10}$$

describes the probability for a walker to stay on the intended path until it quenches the correct fluorophore before time T. For the double-junction circuit in Fig. 4.5(c), we infer that the probability of staying on the intended path *and* reaching the absorbing anchorage within 200

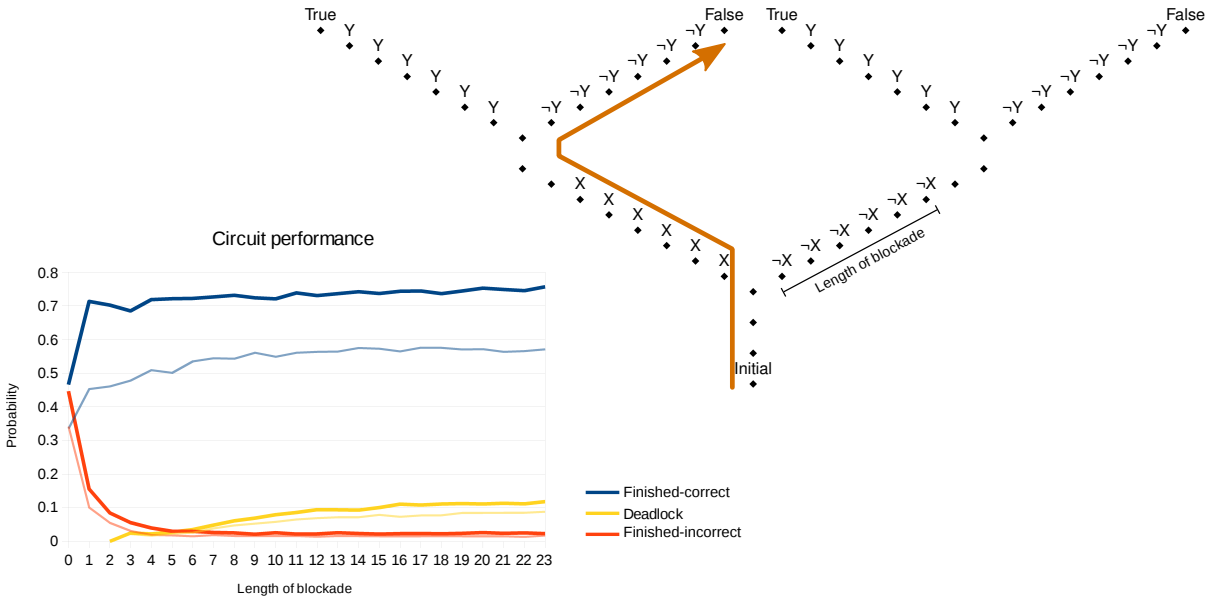


Figure 4.7: Adapted from [4]. Probability of reaching an absorbing anchorage in a double-junction circuit as a function of blockade length. Larger circuits are allowed more time to complete the circuit, and properties are evaluated at $T = 8$ min $\times$ $(7 + 2 \times$ blockade length$)$ for the input $X = \texttt{true}, Y = \texttt{false}$. Faint lines are the same properties given at time $T' = \frac{T}{2}$. Each property is evaluated by simulating 10^4 paths.

minutes is 55% for paths LR and RL, and 58% for paths LL and RR. The walkers on interior paths are indeed more likely to deviate from the intended path than walkers on paths that follow the exterior of the circuit. The double-junction circuit is improved by reducing the probability of leakage. By decreasing the proximity of off-path anchorages and reducing the track length, the proportion of walkers finishing on the correct anchorage is increased (see Table 4.3). The asymmetry between paths (LL, RR vs. LR, RL) also disappears.

Increasing the number of consecutive blockades that form a track guard also results in better performance. Fig. 4.7 shows a redesign of the circuit from Fig. 4.5c; the number of consecutive blockades that constitute a guard is increased from two to six. Guards added beyond the second blocked anchorage are decreasingly effective at improving the probability that the walker arrives at the correct end-node, while the probability of deadlock increases with the depth of the circuit. Deadlock occurs when a walker is isolated on a non-absorbing anchorage with no intact anchorage in range, which can happen when the walker switches direction after stepping over an intact anchorage, as in Fig. 4.6. From a computational standpoint deadlock is undesirable, as it is impossible to differentiate a deadlocked process from a live process. Note that leakage rates similar to that in the original double-junction circuit (see red arrow in Fig. 4.5c) are still present in this circuit.

The expected waiting time before the walker steps is, given the model, equal to $\frac{1}{k}$, where k is the rate of stepping. This means that, for unblocked anchorages on the origami that are

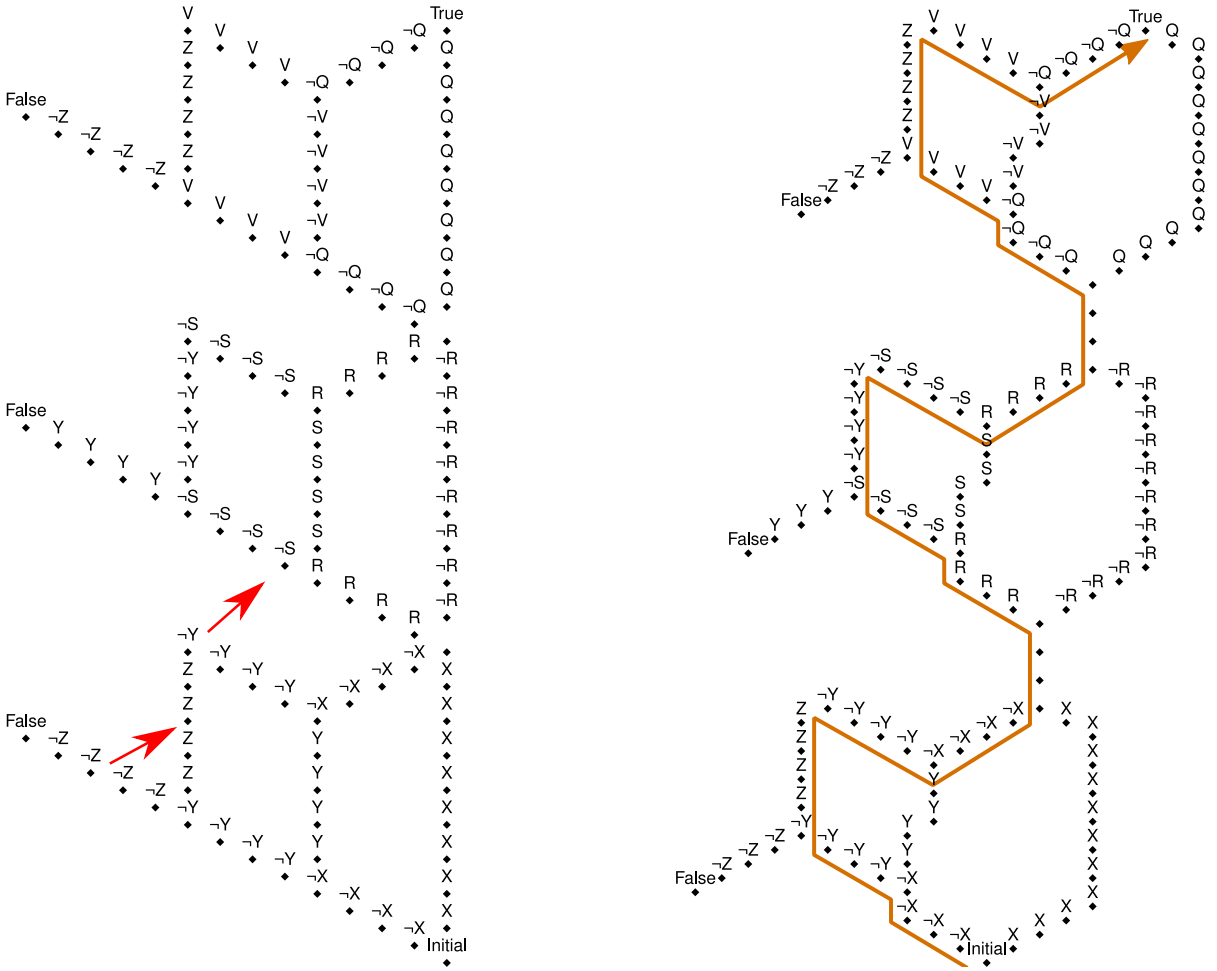


Figure 4.8: Adapted from [4]. Disjunction circuits evaluating $(X \vee Y \vee Z) \wedge (\neg R \vee S \vee \neg Y) \wedge (Q \vee \neg V \vee Z)$. Left: Regular version. Right: Improved version. Red arrows indicate leakage transitions.

immediately adjacent to the walker-anchorage complex, the expected waiting time is $\frac{1}{0.009} \approx 2$ minutes. Fig. 4.7 shows the probability for the walker to finish the computation (correct or incorrect) and the probability for it to deadlock. If the walker were always able to take a step with the base stepping rate $k_s = 0.009$, then we would expect nearly all walkers to finish the computation or deadlock at the given time bound.

In practice, not all of the walkers finish or deadlock, which is due to the walker stepping onto anchorages that are further away (i.e., not immediately adjacent on the origami). It is therefore possible to reach a state where all anchorages that are adjacent to the walker on the origami are cut, but one or more uncut (non-adjacent) anchorages are within range of the walker. In this case, the walker is not (yet) deadlocked; however, the stepping rate to these non-adjacent anchorages is either $k_s/50$ or $k_s/100$, depending on their range. The expected waiting time

before jump occurs can be as high as $\frac{1}{k_s/100} \approx 185$ minutes. By using the property

$$\mathtt{P}_{=?}[\mathtt{F}^{T}(\text{finished} \ \vee \ \text{deadlock})] \tag{4.11}$$

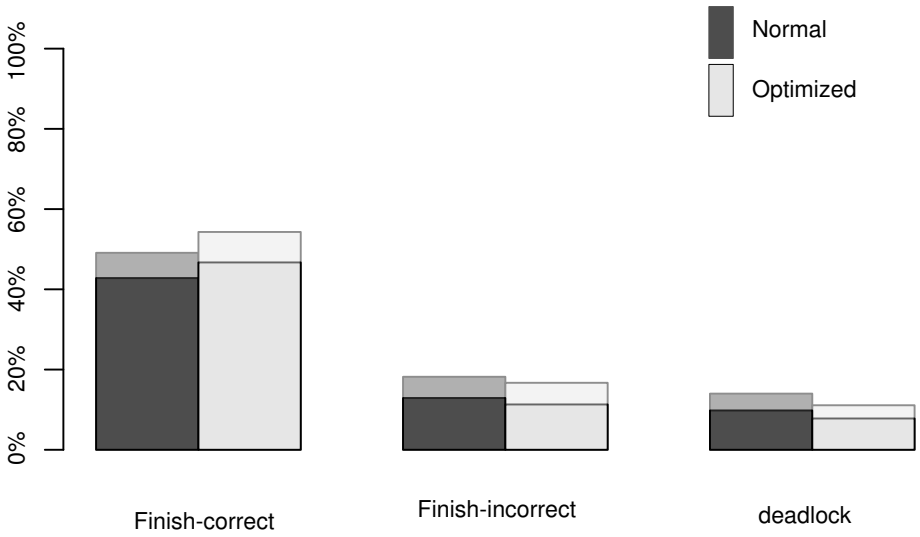
we can compute the probability for the walker to either finish the computation or deadlock at time T. For the circuit in Fig. 4.7, the probability to finish or deadlock when six consecutive blockades are present ($T = 76$ min, Length $= 6$) is equal to 57%. If we remove from the model the ability for the walker to step onto an anchorage that is not adjacent on the origami, then the walker is much more likely to finish or deadlock, as the probability now becomes 90%. This shows that the ability of the walker to step onto anchorages that are not adjacent on the origami degrades the performance of the walker. It is unclear at this point whether the actual walker also suffers from this type of behaviour.

4.4 Design principles for DNA walker circuits

In this section we show how a composable circuit can be optimised, and establish some design principles that increase the reliability of the circuits. We consider a 3CNF circuit with three clauses where each clause is a disjunction over three literals. The first circuit layout in Fig. 4.8 (Left) is a straightforward embedding. However, this layout results in potential leak reactions, some of which are indicated by red arrows. Such leak transitions should be avoided as they increase the probability of error. To minimize the probability of triggering a leak reaction, we apply the following design principles.

- (Principle 1) Increasing the distance between tracks reduces the probability for the walker to deviate from the intended path. Increasing the distance between tracks is accomplished by employing equiangular junctions and by elongating connecting tracks.
- (Principle 2) Increasing the length of the tracks increases the probability for the walker to deadlock. In addition, it increases the expected amount of time until the walker finishes the computation. Therefore, long tracks should be avoided if possible.

These two principles can conflict as increasing the distance between tracks can require that at least one be elongated, thus increasing the rate of deadlock. A pragmatic approach was applied to the design in Fig. 4.8 (Right). The output tracks leading to the false terminals of each clause were made equiangular to two other tracks incident to the common fork gate (principle 1). Furthermore, the connecting tracks between clauses were elongated (principle 1), but only modestly in order to avoid an unnecessary increase in the probability of deadlock (principle 2). We emphasise that the stated design principles are guidelines based on our understanding of the



%	Normal	Optimised	Normal	Optimised
T/ min	252	288	504	576
Finished-correct	42.8	46.7	49.1	54.3
Finished-incorrect	12.9	11.3	18.2	16.7
Deadlock	9.8	7.8	14.0	11.1

Table 4.4: Adapted from [4]. Performance for the disjunction circuits in Fig. 4.8. The time at which the property is evaluated is adjusted proportionally to the size of the circuit, and we evaluate each circuit at two time points. For each of the 128 possible inputs, we simulate 200 paths per property. Top: Barplot of the performance for $T = 252$ and $T = 288$ (solid) and $T = 504$ and $T = 576$ (transparent).

model, and that our software tool was used as a computer aided design (CAD) tool that provides the user with feedback on the performance of circuits. The manually improved circuit is not necessarily optimal, and it is possible that other designs would perform even better. Optimising the circuit does improve its performance, as demonstrated in Table 4.4.

4.5 Conclusion

We have analysed a molecular walker for its applicability as a local computational device. We assert that the molecular walker developed by Wickham et al. [44, 45] is an important demonstration in the development of localized molecular computation, for the following two reasons:

1 The walker allows an efficient embedding of a highly expressive and well-understood class of Boolean formulas, namely the 3-conjunctive normal form, which we demonstrate.

2 The walker enables true localized computation: the walker system operates in an isolated environment, side-stepping any potentially problematic cross-talk that occurs in massively

parallel computation.

Further, we created a probabilistic model of the walker behaviour and applied probabilistic and statistical model checking to identify faulty behaviour. We propose that the reliability and speed of DNA walker circuits is improved by adding blockades and reducing the distance the walker travels. Through a case study of a 9-bit Boolean function we demonstrate how these design rules are applied to optimize performance.

Our analysis benefited from automated model checking methods and the PRISM software. After loading the model to PRISM, a variety of properties were easy to specify and evaluate, and each relevant property was expressed in the temporal logic of CSL (Table 4.1). However, not all results were generated through exhaustive model checking. The 8-anchorage control track was model checked using standard uniformisation. We used fast adaptive uniformisation, a more efficient version of uniformisation, to compute properties for the single-junction circuit. All other results were generated using simulation-based statistical model checking, which is typically less precise than numerical methods based on uniformisation. Section 5.1 includes a discussion of efficiency and precision of these methods.

Chapter 5

Probabilistic model checking for DNA nanotechnology

In this chapter we consider the applicability of probabilistic model checking to models of DNA nanotechnology, and then propose two novel methods to benefit their analysis and synthesis. Both methods are applied to the DNA walker model developed in the previous chapter, which is a (homogeneous) CTMC. In Chapter 6 we discuss a model of DNA origami, which is an inhomogeneous CTMC.

Firstly, we discuss two extensions to the method of uniformisation. The first method, fast adaptive uniformisation (FAU) [124], is a generalisation of standard uniformisation (Def. 3.1.10) and the original contribution of the thesis is an extension of the method to compute rewards (Section 5.1). This section is based on [5, 6]. The FAU method employs a truncated representation of the state-space, neglecting states with low probability, and thus enables the model checking of CTMCs with larger state-spaces. We obtain enhanced performance when computing (reward) properties for the DNA walker model developed in Chapter 4.

Secondly, in Section 5.2 we develop a parameter synthesis method, so that, given a CTMC model with parameters, we can answer precisely for which parameters the model satisfies a CSL property. This is useful when the model parameters are not precisely known, but instead are known to lie within a certain range. The contribution here is the development of parameter synthesis algorithms and a parametric generalisation of the uniformisation method. Using our method we synthesise parameters for the DNA walker model such that performance criteria are met. This section is based on [7, 8].

In Section 5.3 we conclude the chapter.

5.1 Fast adaptive uniformisation

The fast adaptive uniformisation (FAU) method is a generalisation of the adaptive uniformisation method [125] and was first described in [124]. The transient probability at time t is obtained by standard uniformisation as a sum of state distributions after i discrete-stochastic steps, weighted by the probability of observing i jumps in a Poisson process, as in Fig. 3.3. In adaptive uniformisation the discrete distributions are weighted by a birth process instead (see Fig. 3.2) and the discrete process is adjusted accordingly.

After describing the method we discuss how to compute reward properties (Section 5.1.1) and how interval splitting benefits performance (Section 5.1.2). Finally, we demonstrate the advantage of the method over standard uniformisation, using the DNA walker model (Section 5.1.3).

Definition 5.1.1 (*Adaptive uniformisation [125]*)**.** *Let $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ be a CTMC and B a birth process (Def. 3.1.8) with transition rates $q_i \in \mathbb{R}$ subject to $q_i \geq \max_{s \in S} E(s) - \mathbf{R}(s,s)$. Then the transient probabilities π_t over $\mathcal{C}$ are given as*

$$\pi_t = \sum_{i=0}^{\infty} B_{i,t} \overline{\tau}_i \tag{5.1}$$

where $B_{i,t}$ equals the probability to see i jumps (equivalently, being in the i-th state) by time t in the birth process, and the discrete-time probabilities $\overline{\tau}_i$ at iteration i are given as

$$\overline{\tau}_i = \overline{\tau}_{i-1}(\mathbf{I} + \frac{1}{q_i}\mathbf{Q}) \tag{5.2}$$

$$\overline{\tau}_0 = \pi_0. \tag{5.3}$$

The birth process probabilities are approximated using standard uniformisation

$$\hat{B}_{i,t} = \sum_{j=0}^{k_\kappa} \gamma_{j,qt} \tau_j^B(i) \tag{5.4}$$

where q is the uniformisation constant of the birth process, $\gamma_{j,qt}$ is the j-th Poisson probability for a process with parameter qt, $\tau_j^B(i)$ is the probability to be in the i-th state of the discretized birth process at the j-th iteration and k_κ satisfies $\sum_{j=0}^{k_\kappa} \gamma_{j,qt} \geq 1 - \kappa$ for a fixed $\kappa > 0$. An approximation $\tilde{\pi}_t$ to the transient probabilities π_t is then computed as

$$\tilde{\pi}_t = \sum_{i=0}^{k_\epsilon} \hat{B}_{i,t} \overline{\tau}_i \tag{5.5}$$

where k_ϵ *satisfies* $\sum_{i=0}^{k_\epsilon} \hat{B}_{i,t} \geq 1 - \epsilon$ *for fixed* $\epsilon > 0$.

Adaptive uniformisation permits the uniformisation rate q_i to differ between iterations, but as a result the Fox-Glynn algorithm (which computes Poisson probabilities) is no longer applicable. Instead the birth-process probabilities are computed through standard uniformisation (Eq. 5.4). The structure of the birth process is such that $\tau_j^B(i) = 0$ when $i > j$. It permits an iterative computation of $\hat{B}_{i,t}$ that requires only the first $i+1$ elements $q_0, \ldots, q_i$ to be known at each step [145]. Fast adaptive uniformisation reduces the computational cost when compared to adaptive uniformisation by only keeping track of states that have significant probability. The advantage is a reduced cost for the matrix-vector computation in Eq. 5.2 and reduced memory usage. In FAU, the approximation to the transient probabilities is given as

$$\tilde{\pi}'_t = \sum_{i=0}^{k_\epsilon} \hat{B}_{i,t} \tilde{\tau}_i \tag{5.6}$$

for a discrete state probability distribution $\tilde{\tau}_i$ derived from the candidate distribution τ_i^C defined by

$$\tau_i^C = \tilde{\tau}_{i-1}(\mathbf{I} + \frac{1}{q_i}\mathbf{Q}) \tag{5.7}$$

$$\tilde{\tau}_i(s) = \begin{cases} \tau_i^C(s) & \text{if} \quad \tau_i^C(s) \geq \delta \\ 0 & \text{if} \quad \tau_i^C(s) < \delta \end{cases} \tag{5.8}$$

for a fixed probability tolerance $\delta > 0$ and given the initial candidate discrete distribution $\tau_0^C = \pi_0$. The jump probabilities q_i in the birth process need to satisfy

$$q_i \geq \max_{s \in S_i} E(s) - \mathbf{R}(s,s) \tag{5.9}$$

$$S_i = \{s \in S \mid \tilde{\tau}_i(s) > \delta\} \tag{5.10}$$

where S_i is the set of significant states at iteration i. Because the right-hand side in Eq. 5.9 depends on S_i, fixing a single uniformisation rate $q \geq q_i$ ahead of the computation is difficult. The FAU discrete process distribution, $\tilde{\tau}_i$, is typically sparser than that of standard uniformisation, τ_i, resulting in a more efficient method. The solution vector $\tilde{\pi}_t$ is also sparser when compared to standard uniformisation. We measure the saving factor F_{FAU} relative to the size of the entire state space

$$F_{\text{FAU}} = \max_{i \leq k_\kappa} \frac{|S_i|}{|S|} \tag{5.11}$$

and the loss of probability due to Eq. 5.8 and finite summation is measured as:

$$L_{\text{FAU}} = 1 - \sum_{s \in S} \hat{\pi}'_t(s). \tag{5.12}$$

Typically, the savings factor F_{FAU} and loss of probability L_{FAU} are inversely related, and a smaller probability threshold δ or reduced convergence bounds ϵ and κ decrease F_{FAU} and increase L_{FAU}.

5.1.1 Computing rewards with fast adaptive uniformisation

Recall that, given a reward structure $(\rho, \boldsymbol{\iota})$ of state and transition rewards, the CSL-reward operators $\mathtt{R}_{\sim r}[\mathtt{I}^{=t}]$, $\mathtt{R}_{\sim r}[\mathtt{C}^{\leq t}]$ express the instantaneous and cumulative rewards at time t. Historically, the instantaneous rewards are defined to exclude the contribution from transition rewards [96], which we follow in Def. 3.2.4. As a result, the time-derivative of the cumulative rewards is not equal to the instantaneous rewards, that is:

$$\frac{d\mathtt{R}_{=?}[\mathtt{C}^{\leq t}]}{dt} \neq \mathtt{R}_{=?}[\mathtt{I}^{=t}]. \tag{5.13}$$

For a reward structure $(\rho, \boldsymbol{\iota})$ of state and transition rewards, respectively, and transient state probabilities π_t, the instantaneous reward is computed as:

$$\mathtt{R}_{=?}[\mathtt{I}^{=t}] = \sum_{s \in S} \rho(s)\pi_t(s). \tag{5.14}$$

To find the cumulative reward, we must account for the state-transition rewards $\boldsymbol{\iota}$, as well as the expected number of transitions:

$$\mathtt{R}_{=?}[\mathtt{C}^{\leq t}] = \sum_{s \in S} \int_0^t \left(\rho(s)\pi_u(s) + \sum_{s' \in S} \mathbf{R}(s,s')\boldsymbol{\iota}(s,s')\pi_u(s) \right) du \tag{5.15}$$

$$= \sum_{s \in S} \int_0^t \left(\rho(s) + \sum_{s' \in S} \mathbf{R}(s,s')\boldsymbol{\iota}(s,s') \right) \pi_u(s) du \tag{5.16}$$

$$= \sum_{s \in S} \left(\rho(s) + \sum_{s' \in S} \mathbf{R}(s,s')\boldsymbol{\iota}(s,s') \right) \int_0^t \pi_u(s) du. \tag{5.17}$$

For this reason we will focus on the derivation of $\int_0^t \pi_u(s)du$, that is, the expected amount of time the process spends in state s until time t. To demonstrate the correctness of our approach, we need to extend the notion of continuous-time Markov chains. We cast our method in the framework of continuous-time linear propagation models [145, Section 2.3.3 therein].

Definition 5.1.2 (Continuous-time propagation model)**.** *A continuous-time propagation model*

(CTPM) is a tuple $\mathcal{C} = (S, \pi_0, \mathbf{R})$ *where* S *is a countable set of states,* $\pi_0 \colon S \to \mathbb{R}_{\geq 0}$ *is an initialisation vector, and* $\mathbf{R} \colon S \times S \to \mathbb{R}_{\geq 0}$ *is a transition matrix. The propagation vector* π_t *evolves as equivalently to that of the transient probabilities in CTMCs:*

$$\frac{d\pi_t(s)}{dt} = \sum_{s' \in S} \mathbf{R}(s', s) \cdot \pi_t(s') - \sum_{s' \in S} \mathbf{R}(s, s') \cdot \pi_t(s). \tag{5.18}$$

The transition matrix $\mathbf{R}$ assigns a rate $\mathbf{R}(s, s')$ to each pair of states, as for CTMCs, and the initialisation vector π_0 assigns an initial value $\pi_0(s)$ to each state $s \in S$. A CTPM is a CTMC if additionally $\sum_{s \in S} \pi_0(s) = 1$ holds. Adaptive uniformisation also applies[1] to CTPMs [145]. To compute the expected residence time for a CTMC, consider the following construction. We extend its state space by adding time-accumulating states to record the amount of time spent in each state.

Definition 5.1.3 (Time-extended CTPM)**.** *Given a CTMC* $\mathcal{C} = (S, \pi_0, \mathbf{R})$*, the time-extended CTPM is defined as*

$$\mathcal{C}_{\text{ext}} = (S_{\text{ext}}, \pi_{\text{ext},0}, \mathbf{R}_{\text{ext}}) \tag{5.19}$$

where $S_{\text{ext}} = S \cup S_{\text{acc}}$*, so that for each* $s \in S$ *there is a corresponding time-accumulating state* $s_{\text{acc}} \in S_{\text{acc}}$*. The initializing vector is defined accordingly, that is,* $\pi_{\text{ext},0}(s) = \pi_0(s)$ *for* $s \in S$ *and* $\pi_{\text{ext},0}(s) = 0$ *otherwise. The time evolution (also see matrix* $\mathbf{Q}$ *of Def. 3.1.7) is given by*

$$\frac{d\pi_t(s)}{dt} = \begin{cases} \sum_{s' \in S} \mathbf{R}(s', s) \cdot \pi_t(s') - \sum_{s' \in S} \mathbf{R}(s, s') \cdot \pi_t(s) & \text{if } s \in S \\ \pi_t(s') & \text{if } s \in S_{\text{ext}} \text{ and where } s \text{ extends } s' \end{cases} \tag{5.20}$$

The time-extended CTMC is then used to prove the correctness of the following construction.

Theorem 5.1.4 (Residence time)**.** *Consider a CTMC* $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ *and state* $s \in S$*, then*

$$\int_0^t \pi_u(s) du = \sum_{i=0}^{\infty} \Psi_i \overline{\tau}_i(s) \tag{5.21}$$

for $\overline{\tau}_i$ *and* q_i *as in the definition of adaptive uniformisation, Def. 5.1.1, and the mixed birth process probability* $\Psi_i = \frac{1}{q_i} \sum_{n=i+1}^{\infty} B_{n,t}$*.*

Proof. Assume $\mathcal{C}_{\text{ext}} = (S_{\text{ext}}, \pi_{\text{ext},0}, \mathbf{R}_{\text{ext}})$ and let s_{acc} be the time-extended state associated

[1] Note that we did not use the requirement $\sum_{s \in S} \pi_0(s) = 1$ when we derived standard unifomisation (Eq. 3.10).

with s. By the structure of the time-extended CTPM we find

$$\pi_t(s) = \pi_{\text{ext},t}(s) \tag{5.22}$$

$$\int_0^t \pi_u(s)du = \pi_{\text{ext},t}(s_{\text{acc}}). \tag{5.23}$$

Now applying the adaptive uniformisation (Def. 5.1.1) to the CTPM we find:

$$\pi_{\text{ext},t}(s_{\text{acc}}) = \sum_{n=0}^{\infty} B_{n,t}\overline{\tau}_{\text{ext},n}(s_{\text{acc}}) \tag{5.24}$$

where

$$\overline{\tau}_{\text{ext},0}(s_{\text{acc}}) = 0, \tag{5.25}$$

$$\overline{\tau}_{\text{ext},n+1}(s_{\text{acc}}) = \frac{1}{q_n}\overline{\tau}_n(s) + \overline{\tau}_{\text{ext},n}(s_{\text{acc}}) \tag{5.26}$$

and thus

$$\overline{\tau}_{\text{ext},n}(s_{\text{acc}}) = \sum_{i=0}^{n-1} \frac{1}{q_i}\overline{\tau}_i(s). \tag{5.27}$$

Now we apply Eq. 5.27 to Eq. 5.26:

$$\overline{\tau}_{\text{ext},0}(s_{\text{acc}}) = \sum_{n=0}^{\infty} B_{n,t} \sum_{i=0}^{n-1} \frac{1}{q_i}\overline{\tau}_i(s) \tag{5.28}$$

$$= \sum_{n=0}^{\infty}\sum_{i=0}^{n-1} B_{n,t}\frac{1}{q_i}\overline{\tau}_i(s) \tag{5.29}$$

$$= \sum_{i=0}^{\infty}\sum_{n=i+1}^{\infty} B_{n,t}\frac{1}{q_i}\overline{\tau}_i(s) \tag{5.30}$$

$$= \sum_{i=0}^{\infty} \Psi_i\overline{\tau}_i(s). \tag{5.31}$$

□

The result is interpreted as follows: $\overline{\tau}_i(s)$ is the probability to be in state s after i steps in the discrete-time process, and Ψ_i is the probability that at least i jumps occurred in the birth process by time t, multiplied by the expected waiting time before the $(i+1)$-th jump. The time-extended CTPM is only used in the proof and to compute the cumulative rewards, and no extended CTMC is required during computation. The method computes a linear combination of the discrete-time process $\overline{\tau}_i$, similarly to how the transient probability vector π_t is a weighted average of $\overline{\tau}_i$. To compute Ψ_i, only the first $i+1$ uniformisation rates $q_0, q_1, \ldots, q_i$ need to be

known, because

$$q_i \Psi_i = \sum_{n=i+1}^{\infty} B_{n,t} = 1 - \sum_{n=0}^{i} B_{n,t}. \tag{5.32}$$

The infinite summation of Def. 5.1.4 can be approximated by a finite summation in the usual sense: the approximation is a weighted sum of the uniformed process $(\overline{\tau}_i)$ up to some bound k_ϵ (similar to that in Def. 5.1.1). Therefore, computing the residence time has the same computational cost as fast adaptive uniformisation.

5.1.2 Interval splitting

So far we considered FAU for a single time horizon. However, it is often advantageous to consider several smaller time intervals instead, analysing each of them and combining the results, known as interval splitting [146]. This procedure can improve performance and enhance the ability of FAU to deal with stiff models. Interval splitting is known to benefit uniformisation-based methods [146].

Lemma 5.1.5 (Interval splitting)**.** *Given a CTMC $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$, a state reward structure* **r**: $S \to \mathbb{R}_{\geq 0}$*, a time bound $t > 0$, and non-negative interval lengths $t_0, t_1 \ldots t_n$ so that $t = \sum_i t_i$, construct CTMCs*

$$\mathcal{C}_i = (S, \pi_0^i, \mathbf{R}, L) \tag{5.33}$$

where $\pi_0^0 = \pi_0$ and $\pi_0^{i+1} = \pi_{t_i}^i$. Then $\pi_t = \pi_{t_n}^n$ and

$$\int_0^t \pi_u(s) du = \sum_i \int_0^{t_i} \pi_u^i(s) du \tag{5.34}$$

Proof. Trivially $\frac{d\pi_{t-t_i}^i}{dt} = \pi_t \mathbf{Q}$ holds on each sub-interval $t \in [t_i, t_{i+1}]$ by construction of each CTMC $\mathcal{C}_i$ where $\mathbf{Q}$ is the generating matrix of $\mathcal{C}$. The lemma also works when $\mathbf{Q}$ depends on t (when $\mathcal{C}$ is inhomogeneous, see Def. 3.1.13). □

The number of matrix-vector multiplications required, k_ϵ, depends on the number of jumps in the birth process and satisfies $1 - \sum_{i=0}^{N} B_{i,t} < \varepsilon$ for some tolerance ε. The total number of required matrix-vector multiplications can therefore be minimised with respect to a splitting of the interval $[0, t]$. For the FAU method specifically, the cost of the matrix-vector multiplication changes per iteration, and depends on the number of significant states. The cost of solving the birth process B has to be considered as well. The birth process itself is solved using standard uniformisation with a uniformisation rate $q \geq \sup\{q_0, q_1, \ldots, q_n\}$. Sometimes the uniformisation

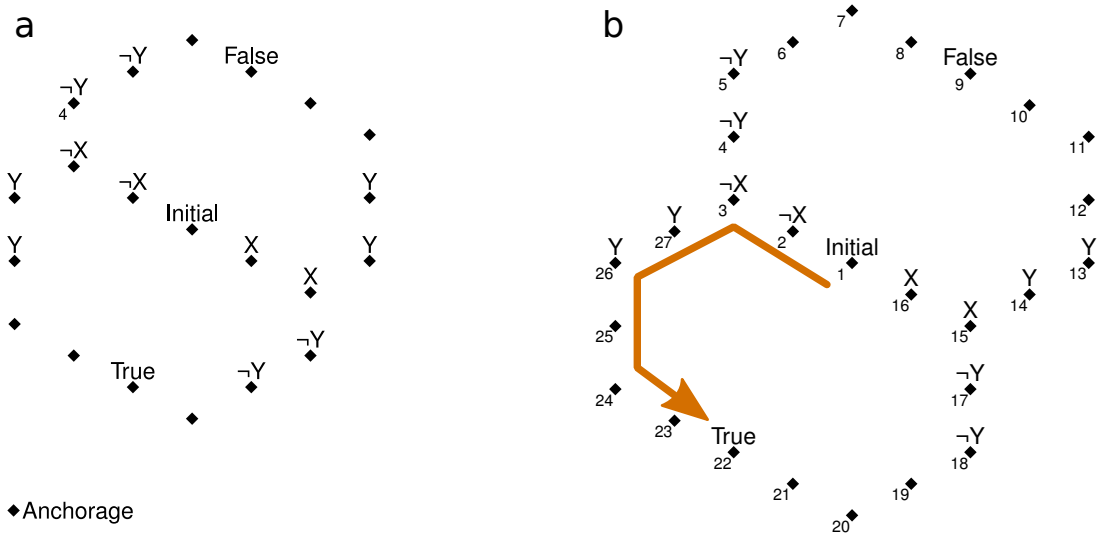


Figure 5.1: From [5, 6]. a) A walker circuit implementing an XOR logic gate. Adding the input $\neg X, Y$ unblocks the anchorages labelled $\neg X$ and Y, directing the walker towards the anchorage labelled True. b) The larger variant of the XOR circuit.

rate for the birth process, q, is significantly larger than the adaptive uniformisation rate at the present discrete time step, so that $q >> q_i$, which increases the number of iterations required to compute the birth process probabilities: this occurs after states with large exit rates are removed from the set of significant states. By splitting the interval we ensure that the standard uniformisation routine used to compute the birth process is not unnecessarily expensive.

5.1.3 Fast adaptive uniformisation applied to walker circuits

We revisit the model of the DNA walker from Chapter 4, which allows the implementation of logic circuits on nanoscale surfaces. In Fig. 5.1 a walker circuit implementing XOR logic is depicted. The walker starts in the Initial position and navigates down each junction [45]. When the walker steps onto an absorbing anchorage, here labelled with True and False, the computation ends. The prior input unblocks certain anchorages, which in turn directs the walker at each junction. Occasionally, the blockade mechanism fails to block an anchorage, which can cause the walker to output the wrong answer. In addition, the walker sometimes steps over blockades or between tracks, which is another source of error. We use the parameter set given in Table 5.1.

We analyse three variants of the XOR-circuit using the FAU method and summarise the results in Table 5.2. The unmodified track, shown in Fig. 5.1a, is "xor", and the suffix "-S" indicates that only one blocker is used instead of two consecutive ones, whereas suffix "-large" indicates the larger track (Fig. 5.1b). (X, Y) or $(X, \neg Y)$ indicates input to the computation, which opens up the blocked anchorages that match the labels of the input. Because the track has a point-symmetry, the results for inputs X, Y and $\neg X, Y$ are identical, as well as for in-

puts $\neg X, \neg\ Y$ and $X, \neg Y$, and we omit the results for these inputs. The unmodified track has 2.9×10^7 states; it can be constructed, but not analysed, with PRISM's standard engines. The larger circuit has 1.9×10^9 states. We compute the expected number of steps (column "Steps", $\mathtt{R}_{=?}[\mathtt{C}^{\leq T}]$) and the probability of walkers reaching the desired anchorage (column "Signal", $\mathtt{P}_{=?}[\mathtt{F}^{T}$ finish-correct]) by time $T = 200$ min. The expected number of steps correlates well with the track layout: when fewer anchorages are blocked ("-S"), the walker takes more steps on average. A larger track also results in more steps taken on average. Column "Blocked", a cumulative reward property, shows how much time the walker spends on anchorages that were supposed to be blocked, and is in line with expectations. When we decrease the threshold cut-off δ (option fau-delta), the number of explored states increases and the lost probability decreases. For $\delta = 10^{-14}$ we find at most $F_{\text{FAU}} = 9\%$ of the total state space concurrently loaded in memory, but only $L_{\text{FAU}} = 1.2 \cdot 10^{-7}$ of the probability is lost.

Parameter	Command line toggle	Value
FAU epsilon	`-fauepsilon`	1e-6
FAU delta	`-faudelta`	1e-8
FAU array threshold	`-fauarraythreshold`	100 (default)
FAU time intervals	`-fauintervals`	1 (default
FAU initial time interval	`-fauinitival`	1.0 (default)

Table 5.1: PRISM settings for the DNA walker case study. The convergence bound for the infinite summation is set equal in both sums, so that $\kappa = \epsilon$ in Def. 5.1.1.

Model	Time (s)	$\max_i \lvert S_i \rvert$	L_{FAU}	Signal	Steps	Blocked (s)
PRISM	*fau-delta 1e-8*					
xor(X, Y)	20	228,803	1.9736E-02	0.6455	7.7696	606.2731
xor$(X, \neg Y)$	22	239,680	2.2587E-02	0.5979	7.5610	659.3715
xor-S-(X, Y)	14	215,544	1.6719E-02	0.5374	8.8363	133.1672
xor-S-$(X, \neg Y)$	15	233,063	1.8775E-02	0.5473	8.4049	146.7377
xor-large-(X, Y)	43	443,584	5.1855E-02	0.5661	9.5020	577.2680
xor-large-$(X, \neg Y)$	45	455,685	5.2995E-02	0.5674	9.4983	567.3420
PRISM	*fau-delta 1e-14, fau-epsilon 1e-9*					
xor(X, Y)	366	2,660,829	1.1838E-07	0.6527	7.8371	627.9572
xor-large-(X, Y)	6923	62,648,566	6.0992E-06	0.5816	9.7201	623.4739

Table 5.2: Adapted from [5, 6]. Fast adaptive uniformisation applied to walker circuits. 'Time' is the amount of time taken for the computation. The maximum of concurrent states is listed ($\max_i \|S_i\|$). Column 'L_{FAU}' shows the maximum percentage of the total statespace loaded in memory.

5.2 Parameter synthesis for parametric Markov chains

Here we describe a novel parameter synthesis method for continuous-time Markov chains. We start with a motivating example. Assuming a hypothetical probabilistic model that automatically resolves conflicts, re-consider the property

The system resolves conflict within 1.0 seconds, with a probability of at least 99.999%.

However, this time around, assume that the system only detects a conflict with some exponential delay, that depends on parameter λ. A natural question now arises: for which range of λ does the property hold?

In Section 5.2.2 we extend the existing method of parametric uniformisation described in [147]. Our novel extension is to allow a wider range of rate functions to be used, and to analyse the numerical stability of the method. The parametric uniformisation is then used as a subroutine to the novel algorithms presented in Section 5.2.3. In Section 5.2.5 we apply the parameter synthesis to the DNA walker model developed in Chapter 4. This work is derived from [7, 8].

The notion of continuous-time Markov chain is extended by allowing transition rates to depend on model parameters [148]. Given a set of variables $K = \{k_0, k_1, \ldots k_{n-1}\}$ we assume that the transition rate $\mathbf{R}(s, s')$ between two states s, s' is now given as a low-degree polynomial over K. The following questions arise naturally:

- For which parameters is the probability of satisfying a CSL property maximized?
- For which range of parameters is the CSL property satisfied?

and we formalize these questions later as Problem 1 and Problem 2. Approximate answers are possible, for example, by computing the satisfaction probability for a finite number of points in the parameter space we can estimate the maximum probability, but there is no guarantee a value near to the maximum is found [148]. We provide a framework which finds the maximum in a precise manner, that is, for Problem 1 the method returns under- and over-approximations of the maximum to within an arbitrarily small precision. For Problem 2, a conservative parameter region is identified. The method relies on being able to find under- and over-approximations of $\frac{\partial \pi_t(s)}{\partial t}$ in each state:

$$\frac{\partial \pi_t(s)}{\partial t} = \sum_{s' \in S} \mathbf{R}(s, s')\pi_t(s') - \sum_{s' \in S} \mathbf{R}(s, s')\pi_t(s) \tag{5.35}$$

This expression is itself a polynomial over K. Finding safe under- and over-approximations is a problem in itself, and relevant to determining which CTMCs permit the use of the method. In this thesis we consider low-degree polynomials, so that approximations to Eq. 5.35 are found analytically. Algebraic expressions can also be found when the rate functions are given as

Hill kinetics. In contrast to previous approaches that support only specific properties (e.g. reachability as in [148]), we consider the time-bounded fragment of CSL, enabling synthesis for an expressive class of properties. Our approach supports arbitrarily precise solutions, which has not been demonstrated before.

We now review some definitions that we require for the analysis of the problem, starting with the definition of the parametric variant of CTMCs and a parametric version of CSL. We provide the problem definition in Section 5.2.1. In Section 5.2.2 we describe how to solve the model checking problem for parametric CTMCs, which includes a description of parametric uniformisation. The iterative algorithms that solve Problem 1 and Problem 2 are described in Section 5.2.3 and Section 5.2.4. To demonstrate their use, we synthesise parameters for the DNA walker in Section 5.2.5.

Definition 5.2.1 (Parametric continuous-time Markov chain)**.** *Assume a set* $K = \{k_0, k_1, \ldots, k_{n-1}\}$ *of model parameters and a parameter space* $\mathcal{P} \subseteq \mathbb{R}^n$ *that is given by bounds* $k_i^{\perp}, k_i^{\top}$ *on each parameter, that is,* $\mathcal{P} = \bigtimes_{k \in K} [k^{\perp}, k^{\top}]$*. A pCTMC over a set* K *of parameters is a tuple* $\mathcal{C}_K = (S, s_0, \mathbf{R}_K, L)$ *where* S *is a countable set of states,* s_0 *an initial state,* $L : S \to 2^{AP}$ *is a labelling function for a set of atomic propositions* AP *and* $\mathbf{R}_K : S \times S \to \mathbb{R}[K]$ *is the parametric rate matrix so that the transition rate between two states is given by a polynomial over the parameters* K*. Given a pCTMC* $\mathcal{C}_K$ *and a parameter space* $\mathcal{P}$*, we denote with* $C_{\mathcal{P}}$ *the set* $\{\mathcal{C}_p \mid p \in \mathcal{P}\}$ *where* $\mathcal{C}_p = (S, \pi_0, \mathbf{R}_p, L)$ *is the CTMC obtained by evaluating the transition matrix* $\mathbf{R}_K$ *at* p *and* $\pi_0 : S \to [0, 1]$ *is the initial distribution where* $\pi_0(s_0) = 1$*.*

We introduce the notion of a satisfaction function that is equal to the probability to satisfy a specific path formula.

Definition 5.2.2 (Satisfaction function)**.** *Let* ϕ *be a CSL path formula,* $\mathcal{C}_{\mathcal{P}}$ *be a pCTMC over a space* $\mathcal{P}$ *and* $s \in S$*. We denote with* $\Lambda_\phi : \mathcal{P} \to S \to [0, 1]$ *the satisfaction function such that* $\Lambda_\phi(p)(s) = P(\omega \in Path(s) \mid \omega \models \phi)$ *in* $\mathcal{C}_p$*.*

5.2.1 Problem definition

We consider parametric continuous-time Markov chain (pCTMC) models of reaction networks that depend on unknown variables in the rate constants and in the initial state. We introduce two parameter synthesis problems for this class of models: the max synthesis problem that, given a CSL path formula ϕ, finds the parameter region where the probability of satisfying ϕ attains its maximum, and the threshold synthesis problem which returns the parameter space where the probability of satisfying ϕ exceeds a predefined threshold r.

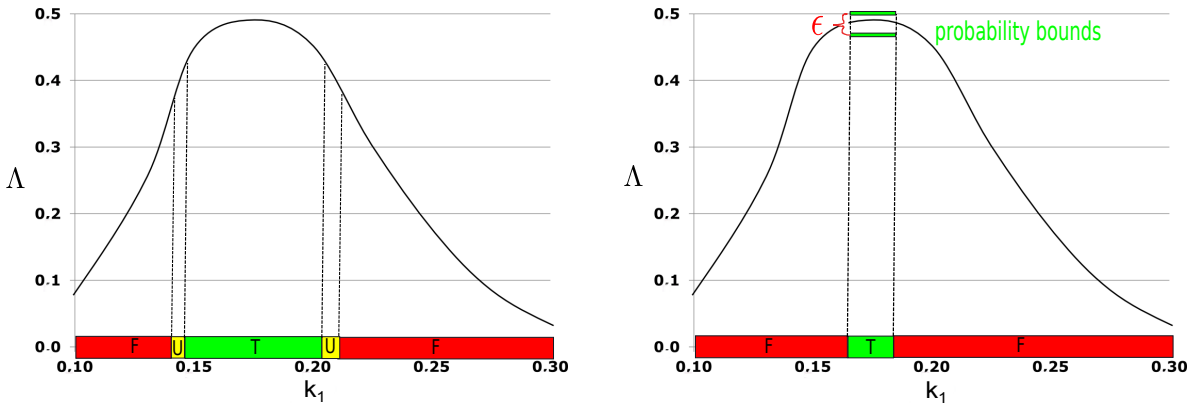


Figure 5.2: The thresholds syntheses (left) and the max synthesis (right) for a sample satisfaction function. Adapted from [7].

Problem 1 (Max Synthesis)**.** *Let $\mathcal{C}_\mathcal{P}$ be a pCTMC over a parameter space $\mathcal{P}$, $\Phi = \mathtt{P}_{=?}[\phi]$ be a CSL formula and $\epsilon > 0$ a fixed tolerance. The max synthesis problem is finding a partition $\{\mathcal{T}, \mathcal{F}\}$ of $\mathcal{P}$ and probability bounds $\Lambda^\perp$, $\Lambda^\top$ such that given an initial state s_0:*

- $\Lambda^\perp - \Lambda^\top \leq \epsilon$;
- $\forall p \in \mathcal{T}.\ \Lambda^\perp \leq \Lambda_\phi(p)(s_0) \leq \Lambda^\top$; *and*
- $\exists p \in \mathcal{T}.\ \forall p' \in \mathcal{F}.\ \Lambda_\phi(p)(s_0) > \Lambda_\phi(p')(s_0)$,

where $\Lambda_\phi(p)$ is the satisfaction function of ϕ on $\mathcal{C}_p$.

Problem 2 (Threshold Synthesis)**.** *Let $\mathcal{C}_\mathcal{P}$ be a pCTMC over a parameter space $\mathcal{P}$, $\Phi = \mathtt{P}_{\geq r}[\phi]$ with $r \in [0,1]$ be a CSL formula and $\varepsilon > 0$ a volume tolerance. The threshold synthesis problem is finding a partition $\{\mathcal{T}, \mathcal{U}, \mathcal{F}\}$ of $\mathcal{P}$, such that given an initial state s_0:*

- $\forall p \in \mathcal{T}.\ \Lambda_\phi(p)(s_0) \geq r$; *and*
- $\forall p \in \mathcal{F}.\ \Lambda_\phi(p)(s_0) < r$; *and*
- $vol(\mathcal{U})/vol(\mathcal{P}) \leq \varepsilon$

where Λ is the satisfaction function of ϕ on $\mathcal{C}_\mathcal{P}$ and $vol(A) = \int_A 1 d\mu$ is the volume of A.

Figure 5.2 illustrates the threshold synthesis regions $\mathcal{T}, \mathcal{U}, \mathcal{F}$ and the max synthesis regions $\mathcal{T}, \mathcal{F}$ for a simple satisfaction function. For max synthesis, the volume of region $\mathcal{T}$ is only indirectly controlled by the tolerance ϵ.

5.2.2 Computing lower and upper probability bounds

This section presents a generalization of the parameter exploration procedure originally introduced in [147]. The procedure takes a pCTMC $\mathcal{C}_\mathcal{P}$ and CSL path formula ϕ, and provides safe under- and over-approximations for the minimal and maximal probability that $\mathcal{C}_\mathcal{P}$ satisfies ϕ,

that is, lower and upper bounds satisfying

$$\Lambda_{\min}(s) \leq \min_{p \in \mathcal{P}} \Lambda_\phi(p)(s) \text{ and}$$
$$\Lambda_{\max}(s) \geq \max_{p \in \mathcal{P}} \Lambda_\phi(p)(s). \tag{5.36}$$

The accuracy of the approximations in Eq. 5.36 is improved by partitioning the parameter space $\mathcal{P}$ into subspaces and re-computing the corresponding bounds, which forms the basis of the synthesis algorithms that we discuss in Section 5.2.3 and Section 5.2.4.

Unnested CSL formulas

We now derive upper and lower bounds for unnested formulas. Bounds for nested CSL formulas are not harder to compute, but do require the correct bound (under or over) to be carried over during the nesting. To this end, we develop an extended version of CSL for pCTMCs in [8]. Our case study, the DNA walker, does not use nested properties, and for the sake of clarity, we omit the parameter synthesis for nested CSL formula from this thesis altogether.

Similarly, reward properties are omitted from this work but are treated in [8]. Finally we note that adaptive uniformisation could be combined with the parameter synthesis method that we present here, but we have not attempted to implement this and this is not featured either here or in [8].

Given an unnested formula ϕ, the model-checking problem for any time-bounded CSL formula reduces to the computation of transient probabilities ([97], example 3.2.3), and a similar reduction is applicable to the computation of lower and upper bounds. We state transient probabilities as given by standard uniformisation and examine the dependency on the model parameters. The probabilities at time t are given by

$$\pi_t = \sum_{i=0}^{\infty} \gamma_{i,qt} \tau_i \tag{5.37}$$

where π_0, $\gamma_{i,qt}$ and k_ϵ are as in Def. 3.10 and $\tau_i = \pi_0 \mathbf{P}_p^i$ is the probability evolution in the discretized process and $\mathbf{P}_p$ is the uniformised matrix obtained from the rate matrix $\mathbf{R}_p$. We derive bounds $\tau_i^{\min}$ and $\tau_i^{\max}$ such that

$$\tau_i^{\min} \leq \min_{p \in \mathcal{P}} \tau_i \quad \text{and} \quad \tau_i^{\max} \geq \max_{p \in \mathcal{P}} \tau_i \tag{5.38}$$

which allow robust approximations to the transient probabilities π_t:

$$\pi_t^{\min}(s) = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt} \tau_i^{\min}(s) \text{ and} \tag{5.39}$$

$$\pi_t^{\max}(s) = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt} \tau_i^{\max}(s) + e_{f-g}. \tag{5.40}$$

where the Fox-Glynn error $e_{f-g} = 1 - \sum_{i=0}^{k_\epsilon} \gamma_{i,qt}$ is due to the bounded summation in the uniformisation. The vector ordering in Eq. 5.38 holds element-wise. The bounds of Eq. 5.36 are obtained from $\pi_t^{\min}, \pi_t^{\max}$ similarly to the computation of $\mathtt{P}_{=?}[\phi]$ from π_t in the non-parametric case (Ex. 3.2.3). To achieve the bounds in Eq. 5.36 we require

$$\tau_{i+1}^{\min}(s) \leq \tau_i^{\min}(s) + \frac{1}{q} \cdot \min_{p \in \mathcal{P}} \ \mathrm{flux}_p(\tau_i^{\min}, s) \tag{5.41}$$

$$\tau_{i+1}^{\max}(s) \geq \tau_i^{\max}(s) + \frac{1}{q} \cdot \max_{p \in \mathcal{P}} \ \mathrm{flux}_p(\tau_i^{\max}, s) \tag{5.42}$$

where

$$\mathrm{flux}_p(\tau_i^{\max}, s) = \underbrace{\sum_{s' \in S} \mathbf{R}_p(s', s) \cdot \tau_i^{\max}(s')}_{\mathrm{inflow}(s)} - \underbrace{\sum_{s' \in S} \mathbf{R}_p(s, s') \cdot \tau_i^{\max}(s)}_{\mathrm{outflow}(s)} \tag{5.43}$$

and q is the uniformisation constant. The extrema for $\mathrm{flux}_p(\tau_i^{\max}, s)$ are found readily when $\mathbf{R}_p$ employs low-degree polynomial expressions. More specifically, when the entries of $\mathbf{R}_p$ are multi-affine polynomials, i.e. multivariate polynomials where each variable has degree at most 1, the maximum in Eq. 5.42 is found by evaluating $\mathrm{inflow}(s) - \mathrm{outflow}(s)$ in the vertices of $\mathcal{P}$ due to the following lemma.

Lemma 5.2.3 (Extrema of multi-affine functions [149, 150])**.** *Let h be a multi-affine function over hyper-rectangular $\mathcal{R}$ and let $V_\mathcal{R}$ be the set of vertices of $\mathcal{R}$. Then,*

$$\min_{p \in \mathcal{R}} h(p) = \min_{p \in V_\mathcal{R}} h(p) \ \textit{and} \ \max_{p \in \mathcal{R}} h(p) = \max_{p \in V_\mathcal{R}} h(p) \tag{5.44}$$

In the multi-affine case, $\tau_{i+1}^{\max}(s)$ is found as

$$\tau_{i+1}^{\max}(s) = \tau_i^{\max}(s) + \frac{1}{q} \max_{p \in V_\mathcal{P}} \mathrm{flux}_p(\tau_i^{\max}, s) \tag{5.45}$$

Provided the rate function $\mathbf{R}_p$ is polynomial of degree d, the solution $\pi_t(s)$ itself can be expressed as a polynomial of degree at most $k_\epsilon d$. A direct attempt to bound the polynomial expression of $\pi_t(s)$ is difficult due to the large number of uniformisation steps, k_ϵ, and previous

approaches in parameter synthesis have provided an approximate solution by sampling the value of π_t over a grid in $\mathcal{P}$ [148], rather than bounding the polynomial itself as in our approach. The computational complexity depends on the rate function but for our settings it has the same asymptotic complexity as standard uniformisation.

Accuracy and convergence

An approximation error is introduced when we compute $\pi_t^{\max}$, or $\pi_t^{\min}$, because the probabilities $\tau_i^{\max}, \tau_{i+1}^{\max}, \tau_{i+2}^{\max}, \ldots$ are locally maximized, allowing different parameter valuations at each step and for each state. This error accumulates at each stage of computing $\tau_i^{\max}$. We examine the numerical convergence of the parametric uniformisation for the multi-affine case where Lemma 5.2.3 applies. Let the parameter space over n variables have volume δ^n, that is, $\mathcal{P} = \bigtimes_{i=1..n}[p_i, p_i + \delta]$ for $\delta > 0$. Fix an arbitrary state s and let the maximizing argument of the transient probability in s be (cf. Eq. 5.37):

$$\hat{p} = \operatorname*{argmax}_{p \in \mathcal{P}} \pi_t(s), \tag{5.46}$$

and let $\hat{\tau}_i = \pi_0 \mathbf{P}_{\hat{p}}^k$. The local error introduced during a single discrete time step is given by

$$\begin{aligned} e_i =& |\hat{\tau}_i(s) - g_s(\hat{\tau}_{i-1})| && (5.47)\\ =& |\hat{\tau}_{i-1} + \frac{1}{q}\mathrm{flux}_{\hat{p}}(\hat{\tau}_{i-1}, s) - \hat{\tau}_{i-1} - \frac{1}{q} \cdot \max_{p \in V_{\mathcal{P}}} \mathrm{flux}_p(\hat{\tau}_{i-1}, s)| && (5.48)\\ =& \frac{1}{q}|\mathrm{flux}_{\hat{p}}(\hat{\tau}_{i-1}, s) - \max_{p \in V_{\mathcal{P}}} \mathrm{flux}_p(\hat{\tau}_{i-1}, s)|. && (5.49) \end{aligned}$$

The error e_i is of order $\mathcal{O}(\delta^n)$, that is, the approximation is first-order in each variable. This also demonstrates the method is consistent, because $e_i \to 0$ as $\delta \to 0$.

Backwards computation

To efficiently find $\Lambda_{\phi,\min}, \Lambda_{\phi,\max}$, a backwards integration is employed as is normal for non-parametric CSL model checking [151]. To illustrate this technique, we give the following scenario. Suppose we have a CTMC $\mathcal{C}$ and time-bounded CSL path formula ϕ and we want to compute

$$\Lambda_p(s) \tag{5.50}$$

for each state $s \in S$ and a fixed parameter assignment p. Then, using the forwards integration, we could set the initial condition $\pi_0(s) = 1$ and $\pi_0(s') = 0$ for all $s' \neq s$, and solve the differential equation for π_T where T is the time-bound of the CSL property, as is described in

Example 3.2.3, and repeat this for each state $s \in S$. Backwards CSL computation improves on this by computing Λ_p for each state directly. In this method, the distribution π_T is fixed and instead π_0 is computed. We further demonstrate this method by extending Example 3.2.3, which we illustrate in Fig. 5.3

For the property

$$\mathtt{P}_{=?}[A\ \mathtt{U}^{[0,T]} B] \tag{5.51}$$

where $[0, T]$ is a non-empty interval of $\mathbb{R}$, the differential equation

$$\frac{d\pi_t}{dt} = \mathbf{Q}\pi_t \text{ with boundary conditions} \tag{5.52}$$

$$\pi_t(s) = \begin{cases} 1 & \text{when } s \models B, \forall t \\ 0 & \text{when } s \not\models B, t = T \end{cases} \tag{5.53}$$

is solved for π_0 given uniformised matrix $\mathbf{Q}$ as in the modified CTMC $\mathcal{C}'$ from Example 3.2.3, and in that case $\pi_0(s)$ is the probability of s to satisfy $A\ \mathtt{U}^{[0,T]} B$. Compared to forwards integration, only one pass of uniformisation is required to compute these probabilities for all states.

Note that, in Eq. 5.52, π_t is no longer a probability mass function, because $\sum_{s \in S} \pi_t(s) = 1$ does not hold in general. Although the backwards computation is a highly practical procedure in time-bounded CSL model checking, it does not change the function or description of parametrised uniformisation (Eq. 5.37 and onwards), where we note that the system of differential equations only differs in the boundary conditions (Eq. 5.52) and we omit further discussion.

5.2.3 Max synthesis

We now describe Algorithm 1, which is used to solve Problem 1, the max synthesis problem, and is based on the repeated use of parametric uniformisation.

Algorithm 1 returns a set of parameters $\mathcal{T} \in \mathcal{P}$ that includes the maximizing argument of the satisfaction function, that is:

$$\operatorname*{argmax}_{p \in \mathcal{P}} \Lambda_\phi(p)(s_0) \in \mathcal{T}. \tag{5.54}$$

Each iteration refines the partition (line 4) of the candidate region $\mathcal{T}$ and excludes the subspaces that do not contain the maximum, until the desired precision is reached (line 3). Let $\cup_i \mathcal{R}_i$ be a partition of $\mathcal{T}$. For each subspace $\mathcal{R}$, the algorithm computes bounds (line 7) on the probability of satisfying ϕ, that is, $\Lambda_{\min}$ and $\Lambda_{\max}$. The algorithm then rules out subspaces by deriving an under-approximation (M) to the maximum satisfaction probability (line 8). If the bound on the

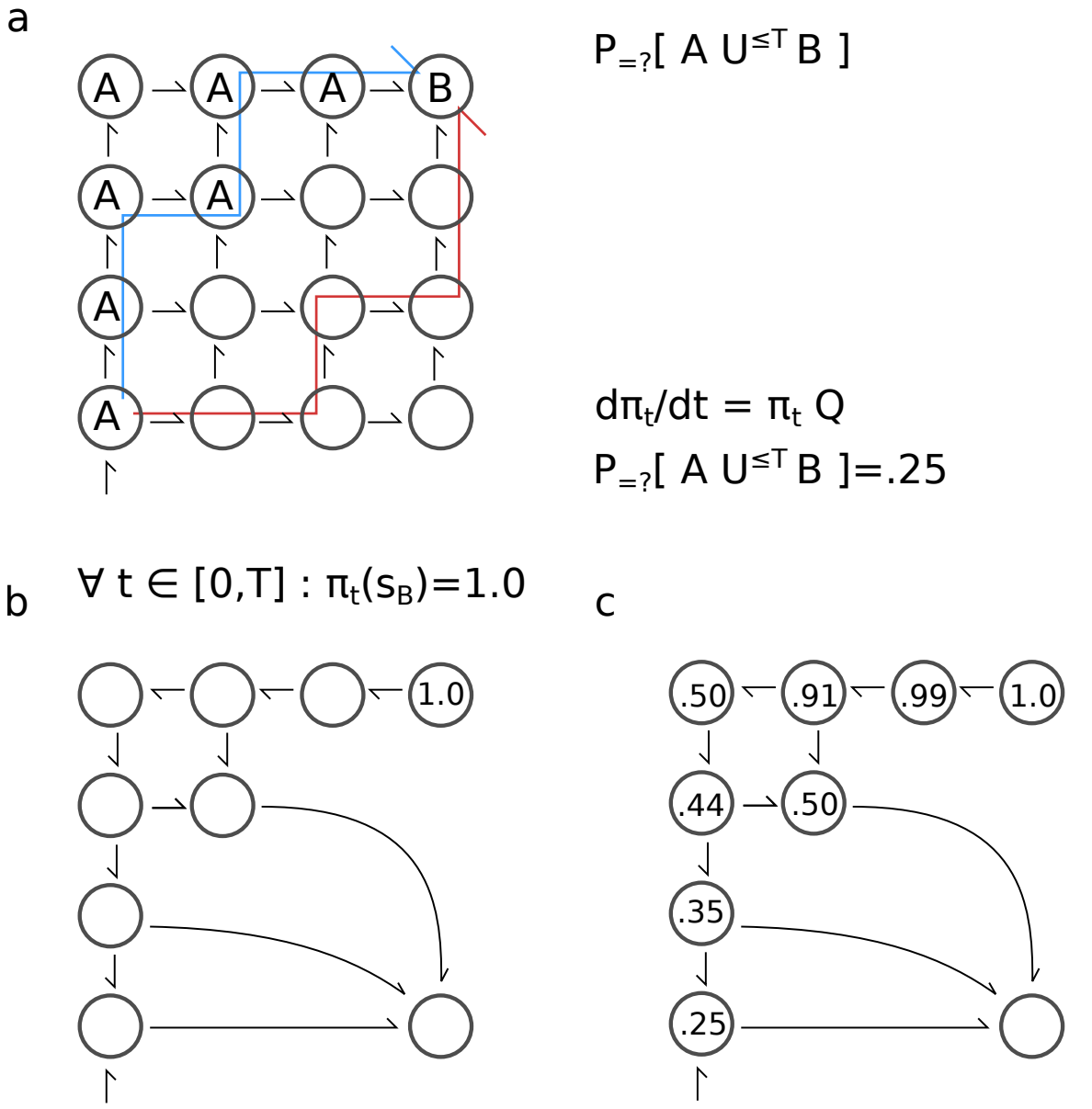


Figure 5.3: Graphical representation backwards integration for $\mathtt{P}_{=?}[A\ \mathtt{U}^{\leq T}B]$. a) As before in Fig 3.4, states are labeled with A and B, and a path satisfies the CSL path property $A\ \mathtt{U}^{\leq T}B$ if the path reaches B without leaving A before time T. In this case, the blue path satisfies the property, but the red path does not. b) The probability for a path, starting in any state, to satisfy $A\ \mathtt{U}^{\leq T}B$, is computed efficiently using by backwards integration. In this case, the ODE is integrated using a different set of boundary conditions, and we solve for π_0 instead of π_T. c) Integrating the probability density backwards from time T, shows that 25% of the paths starting in the bottom left state satisfy the path-property. The probability of satisfying $A\ \mathtt{U}^{\leq T}B$ for paths starting in other states are obtained simultaneously.

maximum probability of satisfying ϕ is lower than the under-approximation M, that is,

$$\Lambda_{\min} < \mathsf{M} \tag{5.55}$$

then the region $\mathcal{R}$ is discarded (line 10). Otherwise, it stays in the set $\mathcal{T}$.

The bound M is derived as follows. In the naive approach, the algorithm uses the maximum over the least bounds in the partition of $\mathcal{T}$, that is,

$$\mathsf{M} = \max\{\Lambda^{\mathcal{R}}_{\min} \mid \mathcal{R} \in \cup_i \mathcal{R}_i\}. \tag{5.56}$$

Let $\overline{\mathcal{R}}$ be the region with highest lower bound. The sampling-based approach improves upon the naive approach by sampling a set of parameters $\{p_1, p_2, \ldots\} \subseteq \overline{\mathcal{R}}$ and taking the highest value of $\Lambda_\phi(p)(s_0)$, that is,

$$\overline{\mathsf{M}} = \max\{\Lambda_\phi(p_i)(s_0) \mid p_i \in \{p_1, p_2, \ldots\}\} \tag{5.57}$$

Each valuation $\Lambda_\phi(p)(s_0)$ is computed through regular CSL model checking, and is equally expensive as computing the bounds on a regular pCTMC. The sampling method results in an improved under-approximation to the maximum of the satisfaction function. As a result the bound rules out more regions, and fewer refinements are required in the next iteration (see Figure 5.4).

5.2.4 Threshold synthesis

We now describe Algorithm 2, which is used to solve Problem 2, the threshold synthesis problem, and is (again) based on repeated use of parametric uniformisation.

Algorithm 2 describes the method to solve the threshold synthesis problem with input formula $\Phi = \mathtt{P}_{\geq r}[\phi]$. The idea is to iteratively refine the undecided parameter subspace $\mathcal{U}$ (line 5) until the termination condition is met (line 4). At each step, we obtain a partition $\cup_i \mathcal{R}_i$ of $\mathcal{U}$. For each subspace $\mathcal{R}_i$, the algorithm computes bounds on the maximal and minimal probability of satisfying ϕ (line 8). The algorithm evaluates if $\Lambda_{\min}$ is above the threshold r, in which case the satisfaction of Φ is guaranteed for the whole region $\mathcal{R}_i$ and thus it is added to $\mathcal{T}$. Otherwise, the algorithm tests whether $\mathcal{R}_i$ can be added to the set $\mathcal{F}$ by checking if $\Lambda_{\max}$ is below the threshold r. If $\mathcal{R}_i$ is neither in $\mathcal{T}$ nor in $\mathcal{F}$, it forms an undecided subspace that retained in the set $\mathcal{U}$. The algorithm terminates when the volume of the undecided subspace is negligible with respect to the volume of the entire parameter space, i.e. $\mathrm{vol}(\mathcal{U})/\mathrm{vol}(\mathcal{P}) \leq \varepsilon$, where ε is the input tolerance. Otherwise, the algorithm continues to the next iteration where $\mathcal{U}$ is further refined.

Algorithm 1 Max Synthesis

Require: pCTMC $\mathcal{C}_{\mathcal{P}}$ over parameter space $\mathcal{P}$, CSL formula $\Phi = P_{=?}[\phi]$ and probability tolerance $\epsilon > 0$

Ensure: $\forall p \in \mathcal{T} : \Lambda^{\perp} \leq \Lambda(p) \leq \Lambda^{\top}$, and $\Lambda^{\top} - \Lambda^{\perp} \leq \epsilon$

$\mathcal{F} \leftarrow \emptyset$
$\mathcal{T} \leftarrow \mathcal{P}$
while $\Lambda^{\top} - \Lambda^{\perp} > \epsilon$ **do**
  $\cup_i \mathcal{R}_i \leftarrow \mathsf{decompose}(\mathcal{T})$
  $\mathcal{T} \leftarrow \emptyset$
  for all $\mathcal{R}_i$ **do**
    $(\Lambda^{\mathcal{R}_i}_{\min}, \Lambda^{\mathcal{R}_i}_{\max}) \leftarrow \mathsf{computeBounds}(\mathcal{C}_{\mathcal{R}_i}, \phi)$
  $\mathsf{M} \leftarrow \mathsf{getMaximalLowerBound}(\cup_i R_i)$
  for all $\mathcal{R}_i$ **do**
    if $\Lambda^{\mathcal{R}_i}_{\max} < \mathsf{M}$ **then**
      $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{R}_i$
    else
      $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{R}_i$
  $\Lambda^{\perp} \leftarrow \min\{\Lambda^{\mathcal{R}}_{\min} \mid \mathcal{R} \in \mathcal{T}\}$
  $\Lambda^{\top} \leftarrow \max\{\Lambda^{\mathcal{R}}_{\max} \mid \mathcal{R} \in \mathcal{T}\}$

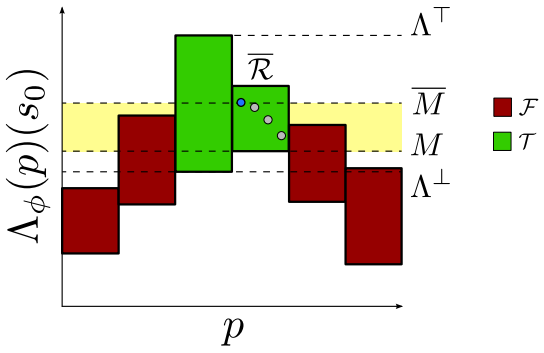


Figure 5.4: Adapted from [7]. Left: Refinement algorithm for max synthesis. Right: The parameter space is divided into several regions, for which the upper and lower bounds of Λ are determined. Regions for which $\Lambda^{\max}$ is less than the lower bound M are discarded. Sampling of $\Lambda(p)$ for random $p \in \overline{\mathcal{R}}$ improves this lower bound to $\overline{\mathsf{M}}$. Without sampling, only the two outer regions would be excluded.

The algorithm benefits from the iterated refinement of $\mathcal{U}$ because the computed bounds approximate the satisfaction function arbitrarily well for non-nested properties, as per Section 5.2.2.

Algorithm 2 Threshold Synthesis

Require: pCTMC $\mathcal{C}_{\mathcal{P}}$ over parameter space $\mathcal{P}$, CSL formula $\Phi = \mathtt{P}_{\geq r}[\phi]$ and volume tolerance $\varepsilon > 0$

Ensure: $\forall p \in \mathcal{T}: \ \Lambda_\phi(p)(s_0) \geq r$, and $\mathrm{vol}(\mathcal{U})/\mathrm{vol}(\mathcal{P}) \leq \varepsilon$

1: $\mathcal{T} \leftarrow \emptyset$
2: $\mathcal{F} \leftarrow \emptyset$
3: $\mathcal{U} \leftarrow \mathcal{P}$
4: **while** $\mathrm{vol}(\mathcal{U})/\mathrm{vol}(\mathcal{P}) > \varepsilon$ **do**
5: $\quad \cup_i \mathcal{R}_i \leftarrow \mathsf{decompose}(\mathcal{U})$
6: $\quad \mathcal{U} \leftarrow \emptyset$
7: $\quad$ **for all** $\mathcal{R}_i$ **do**
8: $\quad\quad (\Lambda_{\min}, \Lambda_{\max}) \leftarrow \mathsf{computeBounds}(\mathcal{C}_{\mathcal{R}_i}, \phi)$
9: $\quad\quad$ **if** $\Lambda_{\min} \geq r$ **then**
10: $\quad\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{R}_i$
11: $\quad\quad$ **else if** $\Lambda_{\max} < r$ **then**
12: $\quad\quad\quad \mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{R}_i$
13: $\quad\quad$ **else**
14: $\quad\quad\quad \mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{R}_i$

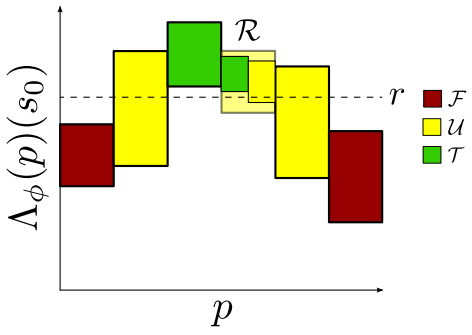


Figure 5.5: Adapted from [7] . Left: Refinement algorithm for threshold synthesis. Right: The parameter space is divided into several regions, for which the upper and lower bounds of Λ are determined. The splitting of $\mathcal{R}$ yields regions in $\mathcal{T}$ and in $\mathcal{U}$.

5.2.5 Synthesis applied to the walker model

We revisit the model of the DNA walker from Chapter 4. The walker model is modified here to allow uncertainty in the stepping rate, and we consider its behaviour over the single-junction circuit of Fig. 4.5(a). Given a distance d between the walker-anchorage complex and an uncut anchorage, and d_a being the distance between consecutive anchorages, the stepping rate k is defined as:

$$k = \begin{cases} k_s & \text{when } d \leq 1.5d_a \\ c \cdot k_s/50 & \text{when } 1.5d_a < d \leq 2.5d_a \\ c \cdot k_s/100 & \text{when } 2.5d_a < d \leq 24\text{nm} \\ 0 & \text{otherwise.} \end{cases} \tag{5.58}$$

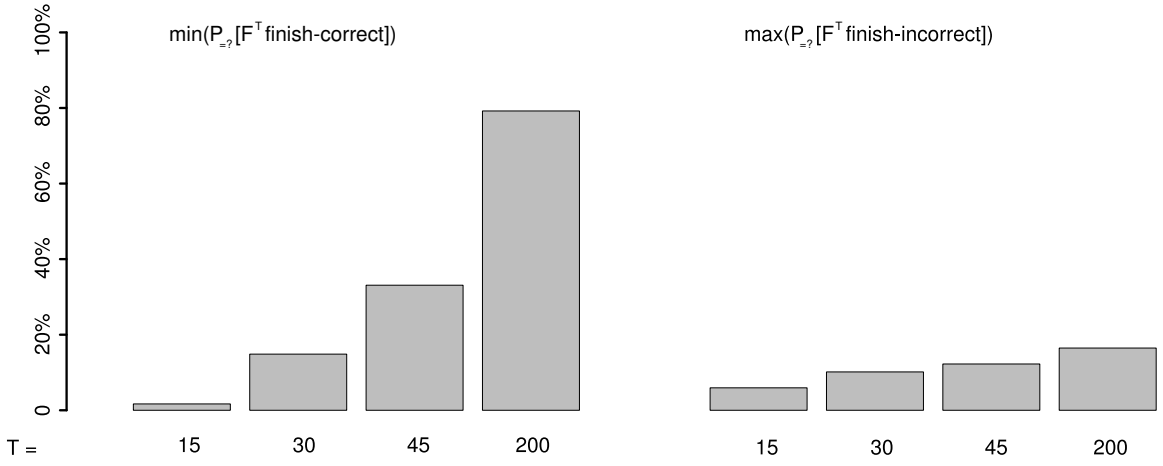
where the base stepping rate $k_s \in [0.005, 0.020]$ is now defined as an interval, as opposed to the original value of 0.009. We have also added factor c for steps between anchorages that are not directly adjacent, but we will assume $c = 1$ for now. The base stepping rate may depend on buffer conditions and temperature, and we want to verify the robustness of the walker with respect to the uncertainty in the value of k_s.

We compute the minimal probability of the walker making it onto the correct final anchorage and the maximum probability of the walker making it onto the incorrect anchorage. We list and plot these probabilities at $T = 15, 30, 45, 200$ minutes in Table 5.3. For time $T = 30, 45, 200$, we note that the walker is robust in the following sense: the least probability for the correct answer is greater than the maximum probability for the wrong answer. For time $T = 15$ this is not the case.

We now consider a property that provides bounds on the ratio between the walker finishing on the correct versus the incorrect anchorage. The rates $c \cdot k_s/50$ and $c \cdot k_s/100$ correspond to the walker stepping onto anchorages that are not directly adjacent, which affects the probability for the walker to end up on the unintended final anchorage. For higher values of c, we expect the walker to end up in the unintended final anchorage more often. We add uncertainty on the value of c, that is, $c \in [0.25, 4]$ and define the performance related property

$$\mathtt{P}_{\geq 0.4}[\mathtt{F}^{30}\ \text{finish-correct}] \wedge \mathtt{P}_{\leq 0.08}[\mathtt{F}^{30}\ \text{finish-incorrect}], \tag{5.59}$$

that is, the probability of the walker to make it onto the correct anchorage is at least 40% by time $T = 30$ min, while the probability for it to make it onto the incorrect anchorage is no greater than 8%. In other words, we require a correct signal of at least 40% and a correct-to-incorrect ratio of at least 5 by time $T = 30$ min. We define a similar property at time $T = 200$ min, this



			Runtime		Subspaces	
Time bound	Min. correct	Max. incorrect	$\emptyset$	Sampling	$\emptyset$	Sampling
$T = 15$	1.68%	5.94%	0.55 s	0.51 s	22	11
$T = 30$	14.86%	10.15%	1.43 s	1.35 s	35	15
$T = 45$	33.10%	12.25%	3.53 s	2.14 s	61	21
$T = 200$	79.21%	16.47%	213.57s	88.97 s	909	329

Table 5.3: Adapted from [7]. Results for max synthesis for $\mathtt{P}_{=?}[\mathtt{F}^{T}$ finish-incorrect] and (similarly defined) min synthesis for $\mathtt{P}_{=?}[\mathtt{F}^{T}$ finish-correct] using $k_s \in [0.005, 0.020], c = 1$ and probability tolerance $\epsilon = 1\%$. The runtime and subspaces are listed for the min-synthesis problem.

time requiring a signal of at least 80%:

$$\mathtt{P}_{\geq 0.8}[\mathtt{F}^{200} \text{ finish-correct}] \wedge \mathtt{P}_{\leq 0.16}[\mathtt{F}^{200} \text{ finish-incorrect}]. \tag{5.60}$$

The synthesized ranges of k_s and c where the properties hold are shown in Figure 5.6. The results agree with the intuition, as in either case the walker cannot deviate too much from the track ($c < 2$). In the case of early measurement (T=30), it is also required that the walker is not too slow ($k_s > 9 \times 10^{-3}$).

5.3 Conclusion

We have developed two new methods for the analysis of CTMC models and applied them to DNA walker model developed in Chapter 4.

We have extended the fast adaptive uniformisation (FAU) method to enable the computation of cumulative reward properties, and tested the performance of the method on DNA walker circuits. The FAU method allowed the model checking of a DNA walker model with 1.9×10^9 states to a precision of $L_{FAU} = 6.1 \times 10^{-6}$ by concurrently loading approx. 63 million states in

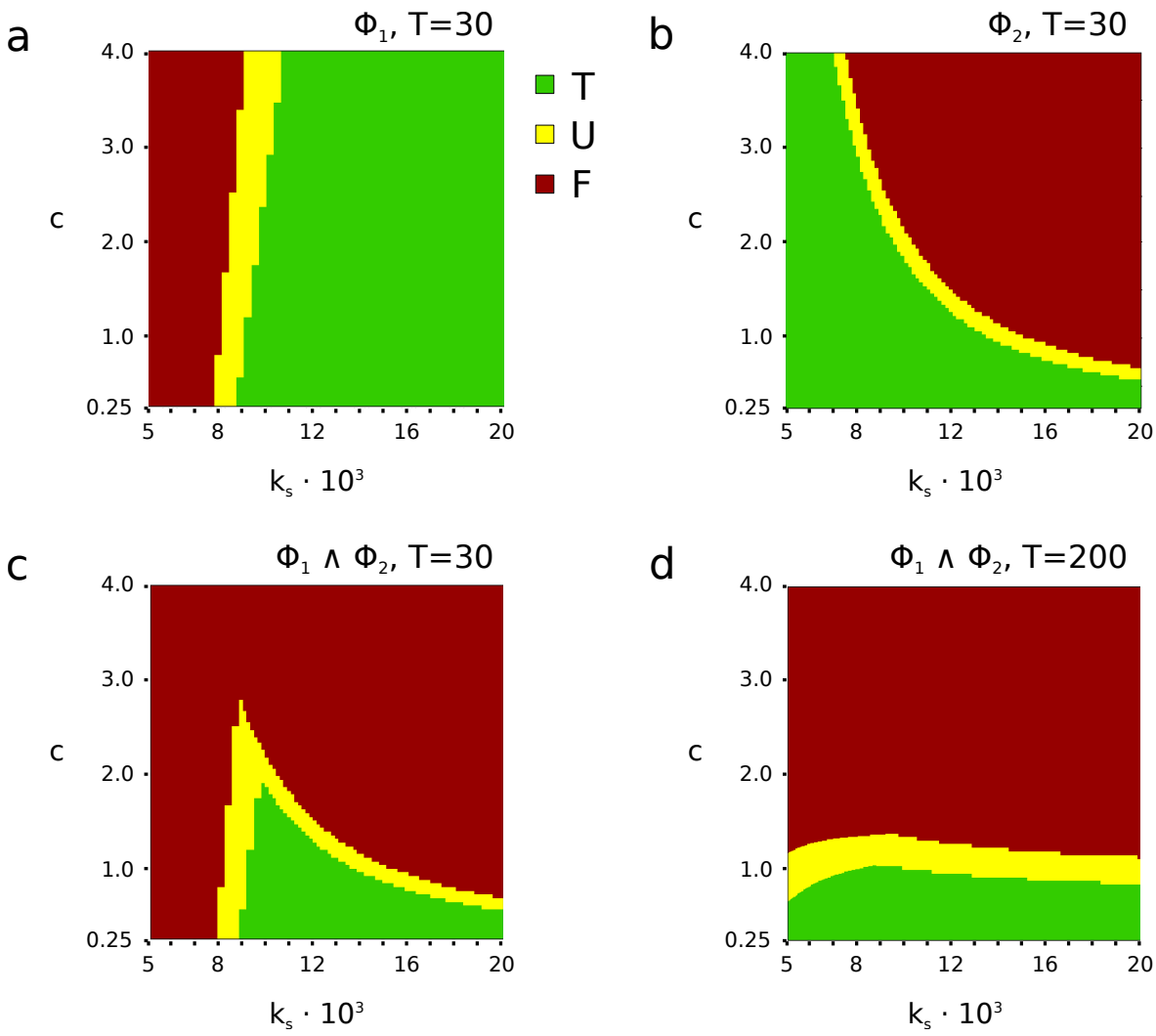


Figure 5.6: Adapted from [7]. We identify parameter sets for which the DNA walker satisfies a performance requirement. Here, c represents a relative intensity for the walker to jump between non-adjacent anchorages, and k_s is the rate of stepping for the walker, as in Eq. 5.58. 'T' (green) indicates parameter regions were the property was satisfied, 'F' (red) where the property was not satisfied and 'U' (yellow) is undecided. In (a) and (b), the undecided region U (yellow) is less than 10% of the total area: this was the stopping criteria for the synthesis algorithm.
In (a), we test if the probability for the walker to finish on the correct final anchorage, within $T = 30$ minutes, is at least 40% ($\Phi_1 = \mathtt{P}_{\geq 0.4}[\mathtt{F}^{30}$ finish-correct], runtime 443.5 s, 2692 subspaces). In (b), we test if the probability for the walker to finish on the incorrect final anchorage, within $T = 30$ minutes, is no larger than 8% ($\Phi_2 = \mathtt{P}_{\leq 0.08}[\mathtt{F}^{30}$ finish-incorrect], runtime 132.3 s, 807 subspaces.) In (c), the overlap of (a) and (b) is plotted, where the green region is the set of parameters that satisfies both requirements ($\Phi_1 \wedge \Phi_2$). To meet both performance requirements, the walker cannot deviate too much from the track ($c < 2$) and cannot be too slow ($k_s > 9{\times}10^{-3}$) In (d), an overlap of similar properties is given for $T = 200$ minutes, where the probability of a correct computation is at least 80%, and the probability for the wrong answer is no larger than 16% ($\mathtt{P}_{\geq 0.8}[\mathtt{F}^{200}$ finish-correct] $\wedge\, \mathtt{P}_{\leq 0.16}[\mathtt{F}^{[200]}$ finish-incorrect], runtime 12.3 h, 47229 subspaces).

memory, which took two hours to compute on workstation hardware[2]. This particular walker circuit embedded an XOR logic gate and had 27 anchorages which, after adding instruction strands, had six remaining blocking strands (Fig. 5.1). The 63 million states represent approximately 3% of the total number of states in the model. The performance of FAU was beyond what could be achieved with the other analysis engines in PRISM, symbolic or explicit. For this particular class of models and our settings, the FAU method typically loads up to 10% of the available states into memory (Table 5.2), and we conclude that FAU performs roughly one order of magnitude better than non-symbolic, explicit-state methods, while the precise benefit strongly depends on the desired accuracy and model.

We also developed a precise method of parameter synthesis for CTMC models, based on a version of parametric uniformisation that we generalized from existing work [147]. This method allows the synthesis of parameter ranges such that CSL properties are guaranteed to hold when the model parameters are within the permitted range. Our method is the first to synthesise parameters for CTMC models through exhaustive analysis of the entire parameter range, and is an improvement over a previous attempt that relied on uniform sampling of the parameters [152]. The method works by bounding the discrete-time process obtained via uniformisation and iteratively refines the parameter space until an acceptable precision is obtained. The runtime of the method strongly depends on the size of the state-space, the property, the number of variables and the width over which the variables are specified. We have applied the synthesis method to the single-junction circuit of Fig. 4.5(a) for which the state-space consists of 5459 states. When uncertainty over 1 parameter was specified, the method computed bounds 329 times for increasingly smaller parameter ranges, which took < 2 minutes (Table 5.3). Another instance involved two unspecified variables and analysis of 47229 subspaces took < 13 hours (Fig. 5.6).

[2]Intel i7-3770 processor with 3.40GHz and 32GB of RAM

Chapter 6

Modelling the self-assembly of DNA origami

DNA origami is a frequently applied technique to create self-assembled nanostructures, and uses short staple strands to fold a longer scaffold strand into shape. Although origami is widely applied, open questions remain. We introduce a model with the aim to gain quantitative insight into the assembly process, and apply it in the next chapter to predict the assembly pathway for a polymorphic tile. We first discuss the self-assembly process and literature in Section 6.1. The remainder of this chapter is devoted to the development of a CTMC model of self-assembly, and is organised as follows. The state space is defined in Section 6.2, and the transition rates are defined in Section 6.3. A model of bulge loop stability is derived through a freely-jointed chain approximation in Section 6.4. The application of the Gillespie algorithm to simulate the self-assembly is described in Section 6.5. Sample rate calculations are given in Section 6.6. We discuss parametrisation and model results in Section 6.7. In Section 6.8 we estimate the minimum required time for the assembly of a DNA origami. Concluding remarks are given in Section 6.9.

Notation: The word 'loop' appears many times in this chapter. Within the context of graphs, 'loop' means a simple cycle, that is, a path that does not visit any vertex or edge more than once, except for the initial vertex. Within the context of DNA, 'loop' refers to a bulge loop.

This chapter is based on a paper that is under review [9].

6.1 The working of DNA origami self-assembly

A design for DNA origami consists of one circular 'scaffold' strand and many shorter 'staple' strands. The staple strands are designed to hybridize to designated domains on the scaffold

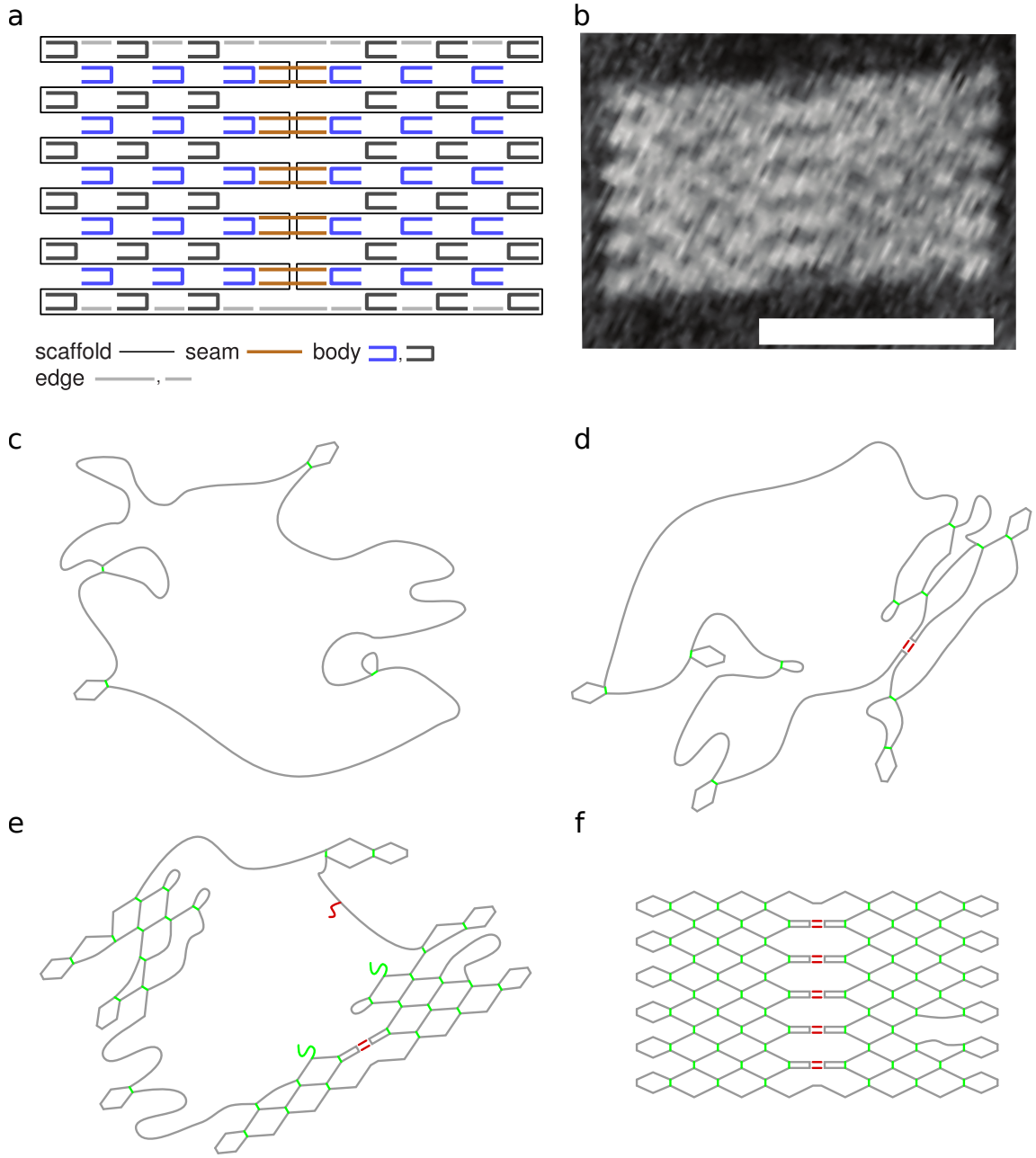


Figure 6.1: DNA origami self-assembly of a rectangular tile. a) Schematic of the origami tile. Staples are complementary to at most two domains on the scaffold and each staple, distinguishable by nucleotide sequence, occurs once in the design. Two-domain staples are shown in blue, black and brown. Single-domain staples are shown in grey. b) AFM imaging of the assembled origami tile with a 50 nm scalebar (imaging by Dr. K.E. Dunn). c) Staples that join two distant domains incur a relatively large entropic penalty compared to staples near the edge, and as a result the latter are earlier to form in the assembly process. d) The incorporation of staples is a reversible process and many intermediate shapes are adopted during higher temperatures. e) As temperatures drops, hybridized domains become more stable and segments of well-formed origami emerge. f) The fully formed origami with a missing staple in the lower right part. The structure is reminiscent of interwoven fabric and within the actual imaging of (b) the five central 'seams' and a diamond-like grid are clearly visible.

through sequence complementarity, and each domain on the scaffold is intended to interact with exactly one type of staple. The scaffold used in the tile design of Fig. 6.1 is 2646 nucleotides long and is a fragment of the pUC19 plasmid. This design features 76 unique two-domain staples and 14 single-domain staples. The single-domain staples, grey in Fig. 6.1, fill the gaps between two-domain staples along the edges. Some of the two-domain staples cross between the central seam of the scaffold routing, and they are called seam staples (brown in Fig. 6.1), while the other two domains staples are called body staples: their distinction is especially relevant in Chapter 7. Domains on the scaffold are either 15 or 16 nucleotides long, with the exception of the two longer domains that are centrally located near the top and bottom of the design, each comprising 32 nucleotides, which are intended to bind with single domain staples. At high temperatures, the hybridization of DNA is unstable and no staples are attached to the scaffold. As the temperature slowly drops, staples start binding to the scaffold, folding it into a pre-determined shape, as in Fig. 6.1. Atomic force microscopy (AFM) is then used to image the tile.

A comparatively non-intrusive method to probe the assembly of DNA origami is by measuring the UV-absorption in the 260 nm spectrum as the temperature decreases. The annealing protocol indicates a temperature gradient in the order of $|dT/dt| = 1.0\,^{\circ}\mathrm{C\ min}^{-1}$. Because double-stranded DNA absorbs less UV light, the amount of absorption goes down as the assembly of the origami progresses. The annealing temperature, T_a, is defined as the temperature at which half the scaffold is double-stranded, and is measured as the temperature at which the absorption is half-way between stable values: a typical experiment requires calibration for drift and the melting temperature may be computed as an average over several runs. The melting temperature, T_m, is defined similarly when the protocol is reversed (causing the origami to disassemble/melt). Assembly was found to exhibit hysteresis [153, 154], that is $T_m - T_a$ is non-zero, indicating that at least either the annealing or melting process occurs out-of-equilibrium. The annealing temperature and yield of multi-layered origami was found to be highly sensitive to staple design [153, 155, 156], and the incorporation of individual staple was found to strongly depend on the presence of surrounding staples [154]. Incubation of structures at fixed temperature suggests that it is the annealing process, rather than the melting process, which occurs out-of-equilibrium and is mainly responsible for the hysteresis [153].

We now discuss two existing models of DNA origami self-assembly [157, 158]. Arbona et al. [157] postulate that the binding of one staple may promote the binding of another by shortening the loop which it must enclose. Similarly to our approach, Arbona et al. [157] use the frequently applied domain-level abstraction (Section 2.3), where hybridization is assumed to occur only between complementary domains, and interaction due to partial sequence overlap is ignored. However, Arbona et al. assume the incorporation of staples is strongly correlated to

increase the tractability of their model: our model does not make such assumptions. The computational model of Arbona et al. results in a deterministic execution, essentially attempting to describe an averaged assembly process. In contrast, our model simulates the assembly of one origami as an explicitly stochastic process over a discrete state-space. Arbona et al. assume nearby base-pairs to directly stabilize other nearby base-pairs, which is suggested to occur due to lateral electrostatic interaction between densely packed helices: such an interaction is not included in our model.

Coarse-grained models of DNA [86, 158], which describe the position and orientation of nucleotides on a continuous scale, are an interesting alternative to our model, because steric constraints are incorporated naturally (we extend our self-assembly model to include these in Chapter 7). In one application, Svaneborg et al. [158] simulate the assembly of a nanostructure consisting of 12 strands of DNA. The model we present here, however, scales better with staple count and we apply it to simulate an origami with 90 staples in this chapter and one with 180 staples in Chapter 7.

6.2 State space

We model the folding of an isolated scaffold surrounded by a large excess of staples, and approach the folding of this origami at the level of domains (see Section 2.3). The model we describe is restricted to designs where staples have at most two domains, such as the origami design in Fig. 6.1, although this approach can be generalized to include staples with more domains [9]. A two-domain staple is considered half-bound when only one domain is hybridized to the scaffold, and fully bound if both domains are bound (Fig. 6.2). We assume that only fully complementary domains can hybridize, ignoring weaker interactions that result from partial sequence complementarity between other pairs of domains. Let the design consist of k staples and let p_i denote the bonding configuration of the scaffold domains which interact with the i-th staple. For single-domain staples we define $p_i \in \{0, 1\}$ where

- 0: a staple is not bound to the scaffold domain;
- 1: a staple is bound to the scaffold domain.

For two-domain staples we have $p_i \in \{00, 10, 01, 11, 12\}$ where:

- 00: no staple bound to either scaffold domain;
- 10: a single staple is bound to the first domain, the second domain is empty;
- 01: a single staple is bound to the second domain, the first domain is empty;
- 11: a single staple is bound to both domains;
- 12: a distinct staple is bound to each domain.

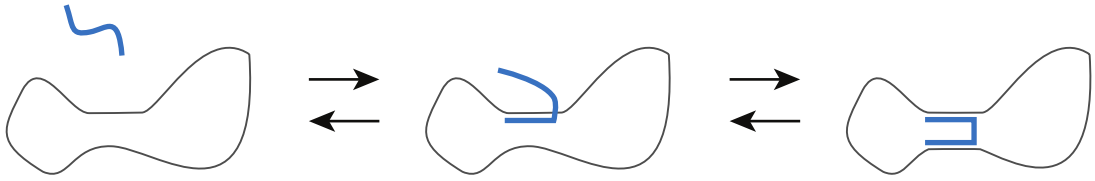


Figure 6.2: Transitions in the model. Some possible bonding configurations of two domains of a scaffold which can bind to a single staple i are shown. From left to right the configurations of this staple denoted $p_i = 00$, $p_i = 01$ (half-bound) and $p_i = 11$ (fully bound).

The state of the scaffold is given by the bonding configuration of the domains, denote $s = (p_0, p_1, \ldots, p_{k-1})$ and let the set of states be S. Then the size of the state space is $|S| = 2^j \times 5^l$, where j is the number of single-domain staples and l the number of two-domain staples.

6.3 Kinetic model and free energy

We develop a CTMC model of the self-assembly of origami given the state space of Section 6.2 and the difference in free energies of partially-folded intermediate states ($\Delta G_{s,s'}$). We assume the out-of-equilibrium process balances according to a Boltzmann distribution and in this approach $\Delta G^0_{s,s'}$ determines the ratio between the rates for the forwards ($s \to s'$) and reverse ($s' \to s$) transitions, and this ratio is given as

$$\frac{\mathbf{R}(s, s')}{\mathbf{R}(s', s)} = \exp\left(\frac{-\Delta G^0_{s,s'}}{RT}\right) \tag{6.1}$$

where R is the gas constant and T is temperature.

As a first approximation to $\Delta G^0_{s,s'}$ we might account for only the free energies of hybridized domains, calculated as if the hybridized duplexes are formed in isolation. This, however, ignores significant interactions between different sections of the origami which depend on the state of the folding. Instead define

$$\Delta G^0_{s,s'} = \Delta G^{\text{duplex}}_{s,s'} + \Delta G^{\text{stack}}_{s,s'} + \Delta G^{\text{shape}}_{s,s'}, \tag{6.2}$$

where $\Delta G^{\text{duplex}}_{s,s'}$ is the contribution from duplex formation as discussed above, and $\Delta G^{\text{stack}}_{s,s'}$ is the contribution from the end-to-end stacking of duplex sections [159], and $\Delta G^{\text{shape}}_{s,s'}$ is the contribution from the entropic costs of scaffold loop formation.The approximations for the various contributions to $\Delta G^0_{s,s'}$ are the central topic of this section and the next (Section 6.4). We now discuss the functional expressions for $\mathbf{R}(s, s')$ based on Eq. 6.1 for each of the possible transitions in the model.

Consider an isolated origami in a partially folded state s_{00} with $p_i = 00$, and let staple p

bind to the scaffold by a single domain, resulting in a state s_{01} with $p_i = 01$ (Fig. 6.2). This transition rate is assumed analogous to isolated duplex formation (Section 2.5) and is given by

$$\mathbf{R}(s_{00}, s_{01}) = k_+[p] \tag{6.3}$$

where $[p]$ is the concentration of the staple. Although binding rates are known to be weakly dependent on duplex stability compared to unbinding rates [23], we assume k_+ is independent of temperature, domain sequence, and folding state, and we fix $k_+ = 10^6\mathrm{M}^{-1}\mathrm{s}^{-1}$ as a reasonable first approximation [82]. The rate $\mathbf{R}(s_{01}, s_{00})$ for the reverse reaction is then given by a Boltzmann equation as:

$$\mathbf{R}(s_{01}, s_{00}) = k_+ \exp\left(\frac{\Delta G^0_{s_{00},s_{01}}}{RT}\right) \times \mathrm{M}. \tag{6.4}$$

were $\Delta G^0_{s,s'}$ is the difference between the free energies of partially-folded intermediate states $s, s' \in S$ and M is the molar unit. An equivalent approach is used for any transition involving the binding of the first arm of a staple. The binding and unbinding transitions of the second domain of staple p to form s_{11} with $p_i = 11$ are associated with a similar thermodynamic constraint:

$$\frac{\mathbf{R}(s_{01}, s_{11})}{\mathbf{R}(s_{11}, s_{01})} = \exp\left(\frac{-\Delta G^0_{s_{01},s_{11}}}{RT}\right). \tag{6.5}$$

To resolve the ambiguity in the absolute values of $\mathbf{R}(s_{01}, s_{11})$ and $\mathbf{R}(s_{11}, s_{01})$, we make the assumption that the unbinding rate $\mathbf{R}(s_{11}, s_{01})$ is equal to the temperature- and sequence-dependent unbinding rate for the corresponding isolated duplex:

$$\mathbf{R}(s_{11}, s_{01}) = k_+ \exp\left(\frac{\Delta G^{\mathrm{duplex}}_{s_{01},s_{11}} + \Delta G^{\mathrm{stack}}_{s_{01},s_{11}}}{RT}\right) \times \mathrm{M}. \tag{6.6}$$

Then combining Eq. 6.5 and Eq. 6.6, we find

$$\mathbf{R}(s_{01}, s_{11}) = k_+ \exp\left(\frac{-\Delta G^{\mathrm{shape}}_{s_{01},s_{11}}}{RT}\right) \times \mathrm{M}. \tag{6.7}$$

Higher $\Delta G^{\mathrm{shape}}_{s_{01},s_{11}}$ corresponds to more demanding constraints associated with the binding of the second arm within the origami. In this approach these constraints are manifested as a slower binding rate for the second domain and an equivalent approach is used for any transition involving the binding of the second arm of a staple.

6.3.1 Duplex free energy

$\Delta G^{\text{duplex}}_{s,s'}$ is calculated using the well-established SantaLucia parametrization of the nearest-neighbour model of DNA thermodynamics [24, 26]. The contribution is computed as

$$\Delta G^{\text{duplex}}_{s,s'} = \Delta G^{\text{duplex}}_{s'} - \Delta G^{\text{duplex}}_{s} \tag{6.8}$$

where $\Delta G^{\text{duplex}}_{s}$ is the sum of free energy contributions of each hybridized domain in s, computed through the SantaLucia parameters for the nearest-neighbour model of duplex stability. We assume buffer conditions of 40mM [Tris] and 12.5mM [Mg^{2+}] and apply an additional entropic penalty to duplex formation depending on these ionic conditions [24, 78, 160]:

$$\Delta S^{0,\text{salt}} = 0.368 \times \frac{N}{2} \times \ln\left(\frac{1}{2}[\text{Tris}] + 3.3[Mg^{2+}]^{1/2}\right), \tag{6.9}$$

in which N is the number of phosphates in a duplex.

6.3.2 Stacking free energy at nicks

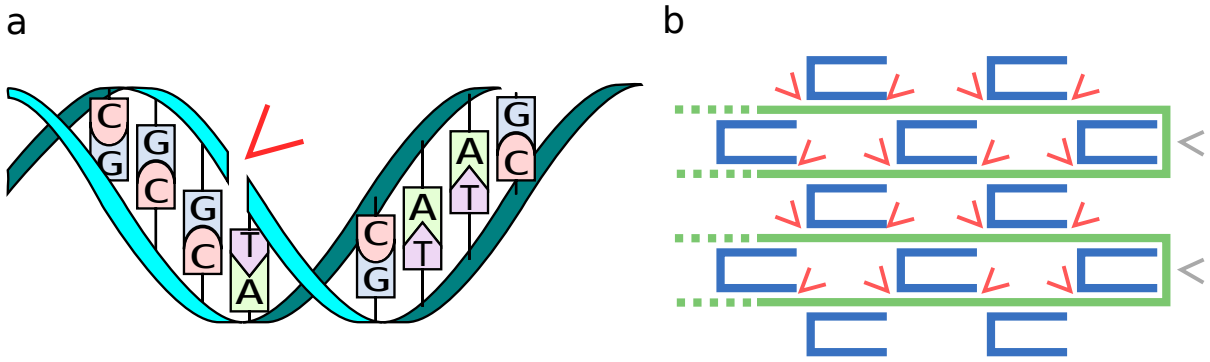


Figure 6.3: a) Nicks occur in DNA when a phosphate backbone is missing in otherwise double-stranded DNA, and result in stacking interaction between the two sections (Creative Commons, M. P. Ball, modified). b) Stacking at nicks occurs in fully folded origami when domains align, and are indicated with red angle brackets. Angle brackets in grey: nick-stacking interactions that are included in the model, although the involved domains cannot physically stack.

The stacking of bases across a nick is a stabilizing interaction [159], and it is reasonable to assume that a similar effect occurs when staples bind to adjacent domains of the origami, as depicted in Fig. 6.3. The contribution is computed as

$$\Delta G^{\text{stack}}_{s,s'} = \Delta G^{\text{stack}}_{s'} - \Delta G^{\text{stack}}_{s} \tag{6.10}$$

where $\Delta G^{\text{stack}}_{s}$ is the sum of free energy contributions from stacking interactions across nicks in

a state s.

In the model, stacking across nicks occurs whenever two adjacent domains on the scaffold are both double-stranded, and using this approach we neglect geometrical circumstances, such as bends in the scaffold routing, that prevent the stacking of two neighbouring domains in the physical origami. For convenience, we take

$$\Delta G^{\mathrm{stack}} = n\langle \Delta G^{\mathrm{bp}} \rangle. \tag{6.11}$$

where $\langle \Delta G^{\mathrm{bp}} \rangle$ is the sequence-averaged free-energy gain per base pair in the nearest-neighbour model. We treat n as an adjustable parameter of the model, and explore the consequences of varying n in Section 6.7.

6.3.3 Scaffold shape free energy

In order to establish an expression for

$$\Delta G^{\mathrm{shape}}_{s,s'} = \Delta G^{\mathrm{shape}}_{s'} - \Delta G^{\mathrm{shape}}_{s} \tag{6.12}$$

we develop a novel approximation to establish $\Delta G^{\mathrm{shape}}_{s}$, that is, the entropic cost of restricting the scaffold to a particular shape. The presence of m fully bound two-domain staples puts m additional constraints on the conformation of the origami, because each crossover between two staple domains restricts the orientation and movement of those domains. This reasoning suggests

$$\Delta G^{\mathrm{shape}}_{s} = G^{\mathrm{shape}}_{s} - G^{\mathrm{shape}}_{\mathrm{null}} = \sum_{j \in L(s)} \Delta G^{\mathrm{loop}}_{j} \tag{6.13}$$

where $L(s)$ is a set of loops present in the partially folded structure s, and $G^{\mathrm{shape}}_{\mathrm{null}}$ is the conformational free energy for an origami with no staples. The idea here is to identify one loop for each of the m staples, and we discuss the exact form of $\Delta G^{\mathrm{loop}}_{j}$ only in the next section.

We now outline which loops are included in the set $L(s)$. We interpret the partially folded origami as a graph, such that each fully-bound two-domain staple contributes two vertices and an edge. We identify loops on this graph, by finding cyclic paths that do not traverse the same edge twice. The total number of such loops (or simple cycles) grows exponentially with the number of bound staples.

Intuitively, each additional staple that is added to the origami restricts the conformational freedom of the scaffold further, and our model aims to take this into account. It is difficult, however, to create a meaningful map from the exponentially many possible loops, onto a defi-

nition of entropic cost. Also, keeping track of exponentially many loops is unpractical from a computational perspective. We now introduce two solutions to this problem, which we name the *local* model and *global model*. In the local model, we only consider the entropic cost of forming the smallest identifiable loop. In the global model, we keep track of only the loops that make up the faces of a planar graph (that is, $m+1$ loops at a time).

Local and global model

Given a scaffold with m fully-bound two domain staples, it is not directly clear how to chose m loops that give the most physical representation of G_s^{shape}. We offer two solutions to this problem, namely a local and global approach. The local approach applies to both planar and non-planar graphs, but is not thermodynamically well-defined. The global approach is thermodynamically consistent, but currently does not apply to non-planar graphs. The DNA origami considered in this chapter, the tile of Fig. 6.1, has a state space that consists exclusively of planar graphs, and for that reason both methods apply. In Chapter 7 we consider the folding pathways of a polymorphic tile, which does not permit planar embeddings for all states, and in that chapter we exclusively employ the local model.

The two approaches are compared in Section 6.7.3 to explore whether the local model is suited to study origami when the global approach is not applicable. Before we discuss the specifics of computing $\Delta G_{s,s'}^{\text{shape}}$, we briefly discuss our assumptions so far and why the global model is developed.

Justification of the model so far

At this point, we note that both models are coarse approximations to the actual entropic penalty of scaffold folding, e.g. when in Eq. 6.13 we resolve that

$$\Delta G_s^{\text{shape}} = \sum_{j \in L(s)} \Delta G_j^{\text{loop}} \tag{6.14}$$

then note this is simply an approximation. The description of budge loop formation is fitting for staple binding occurring at early stages of the folding, but fails to take the increase in rigidity into account that exists in nearly-completed origami. A major benefit of our approach, however, is that the free energy cost of bulge loop formation can be understood analytically, and the expression can be calibrated to experimental estimates [26]. In the future, a more accurate expression of $\Delta G_s^{\text{shape}}$ can be used instead.

Our model fails to fully account for steric constraints, for example, the possibility of a scaffold entanglement is not present in our approach. Similarly, the model only allows considers

hybridization of fully complementary domains, and does not take the secondary structure of the scaffold into account. In light of these assumptions, our form of $\Delta G_s^{\text{shape}}$ should be seen as a practical first approximation.

The local model provides, given a half-bound staple, a direct estimate of the local concentration of the opposing domain. Later in this section, we will find that the local model compares favorably to experimental observations (Fig. 6.12) and, in Chapter 7 that it simulates the self-assembly process to a degree accurate enough to enable the rational design of origami folding pathways. So why bother with the global model?

As mentioned, the local model is not energy-consistent (Fig. 6.4), and we wish to validate its results by comparing it against an energy-consistent model. The requirements for this second model are:

1 Computing the transitions should be computationally inexpensive, and

2 The model should be as similar as possible to the local model, but not suffer from the behaviour of Fig. 6.4.

To meet these requirements, the global model was developed.

Local model

The local model identifies the cost of the smallest loop that forms or is disrupted during each transition, and we neglect the effects of the transition on other loops in the graph. Thus, in the local model,

$$\Delta G_{s,s'}^{\text{shape}} = \begin{cases} \Delta G_{\text{min}}^{\text{loop}} & \text{if a loop forms under } s \to s', \\ -\Delta G_{\text{min}}^{\text{loop}} & \text{if a loop forms under } s' \to s, \\ 0 & \text{otherwise.} \end{cases} \tag{6.15}$$

The local approach does not lead to a well-defined G_s^{shape} for each configuration (see Fig. 6.4), but is computationally less demanding than the global version as it does not require explicit enumeration of all faces in a graph.

Global model

For a given planar embedding, it is easy to identify faces (and hence loops) within an origami, as illustrated in Fig. 6.5. However, a given graph has multiple possible planar embeddings, as is also shown in Fig. 6.5. To resolve this ambiguity, we specify whether the edge associated with each staple lies on the inside or on the outside of the scaffold based on the intended origami

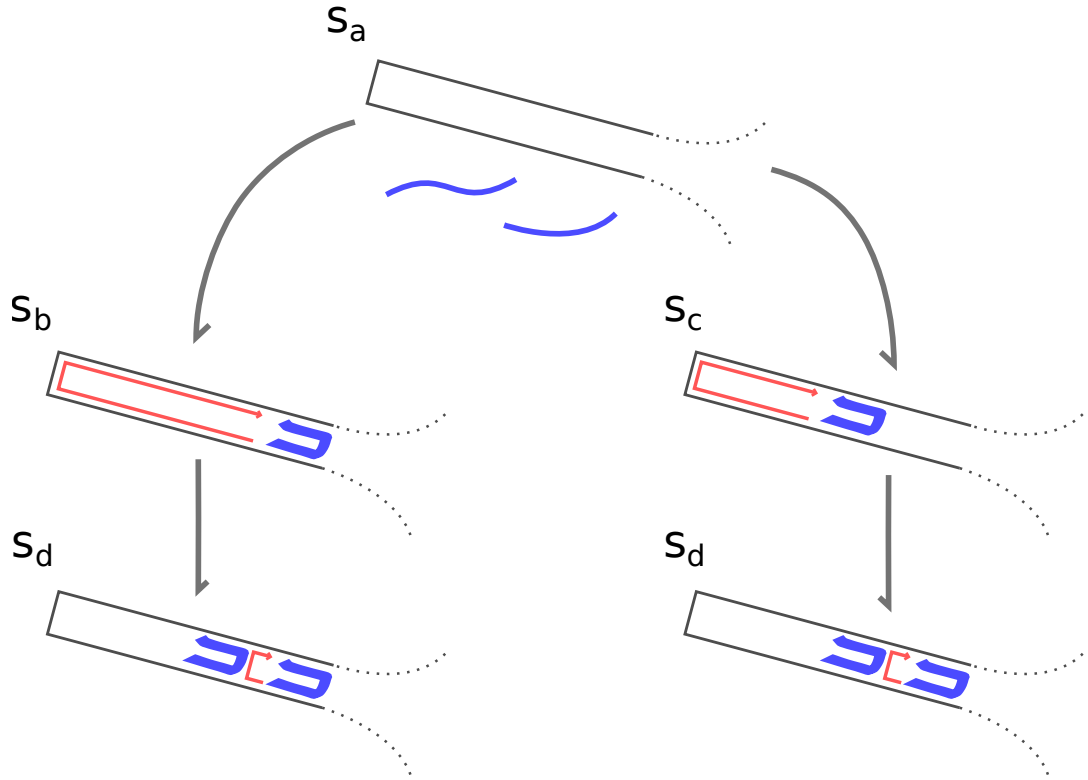


Figure 6.4: Two staples (blue) hybridize to an otherwise empty scaffold (green). Thermodynamic inconsistency occurs in the local model because only the smallest loop (in red) is identified during each transition. Because the change in free energy depends on the length of the loop, we find in this case $\Delta G^{\text{shape}}_{s_a,s_b} + \Delta G^{\text{shape}}_{s_b,s_d} \neq \Delta G^{\text{shape}}_{s_a,s_c} + \Delta G^{\text{shape}}_{s_c,s_d}$.

structure. Viewing the simple origami design of Fig. 6.1a as a planar structure, we identify that some body staples connect on the exterior of the scaffold, while others join inside the interior (blue versus black and brown in Fig. 6.1), and this assignment makes the planar embedding of the graph unambiguous at any stage of folding. A graph with m two-domain staples has $m+2$ faces, and the empty scaffold is treated as if both the initial inner and outer face contribute a separate loop constraint: this could be corrected for at a later point: however, given the large number of loops in the fully folded structure, this is only a minor inaccuracy.

In our global approach, we calculate $\Delta G^{\text{shape}}_{s,s'}$ as described so far. This approach is 'global' because it accounts for the consequences of an incoming (or outgoing) staple for all present loops, in addition to the contribution of a newly added or removed loop. If the ΔG^{loop}_j values were not updated, the resulting dynamics would not form an energy-consistent model. A model is energy-inconsistent if some states s_a, s_b, s_c, s_d exist such that

$$\Delta G_{s_a,s_b} + \Delta G_{s_b,s_d} \neq \Delta G_{s_a,s_c} + \Delta G_{s_c,s_d} \tag{6.16}$$

because transitivity applies to the difference in free energy, so both sides of the equation should

equate to $\Delta G_{s_a,s_d}$. The global model is energy-consistent, i.e., it follows from Eq. 6.8, Eq. 6.10 and Eq. 6.13 that

$$\Delta G(s, s') = \Delta G_{s'} - \Delta G_s. \tag{6.17}$$

6.4 Free energy of DNA bulge loops

Both the local and global model require a definition of the free-energy cost ΔG^{loop} associated with the formation of a single loop. We consider the simple case of isolated bulge loop formation, as in Fig 6.6, and derive an estimate for the free energy cost of loop formation as a function of length and composition of the loop, allowing the loop to contain both single- and double-stranded domains. Models of loop formation of purely single-stranded chains exist in the literature [24], to which we fit our model. Then, in the self-assembly model, we assume the cost of loop formation ΔG^{loop} is equal to that of isolated bulge loop formation for chains of equal composition. The model permits a free variable in the form of the 'loop exponent' that arises naturally in our derivation. Examples of rate calculation are given in Section 6.6.

Given the bulge loop formation as in Fig. 6.6, we label the two states h and f for the half-bound and fully bound state. As for the states s_{01}, s_{11} in the origami model, which also represent half-bound and fully bound staple bindings (respectively), the balance of rates follows the Boltzmann equation:

$$\frac{\mathbf{R}(h, f)}{\mathbf{R}(f, h)} = \exp\left(\frac{-\Delta G^{\text{duplex}}_{h,f} - \Delta G^{\text{stack}}_{h,f} - \Delta G^{\text{loop}}_{h,f}}{RT}\right) \tag{6.18}$$

The unbinding rate of the domain is taken as equal to the unbinding rate in ordinary bimolecular duplex formation, as is the case in the origami model:

$$\mathbf{R}(f, h) = k_+ \exp\left(\frac{\Delta G^{\text{duplex}}_{h,f} + \Delta G^{\text{stack}}_{h,f}}{RT}\right) \times \text{M}. \tag{6.19}$$

The combination of Eq. 6.18 and 6.19 results in

$$\mathbf{R}(h, f) = k_+ \exp\left(\frac{-\Delta G^{\text{loop}}_{h,f}}{RT}\right) \times \text{M}. \tag{6.20}$$

Our approach is to offer a direct approximation of $\mathbf{R}(h, f)$ by treating it as a bimolecular reaction, where the effective concentration of the opposing domain is approximated using a freely-jointed chain model. This then results in a functional definition of ΔG^{loop} that depends on the length

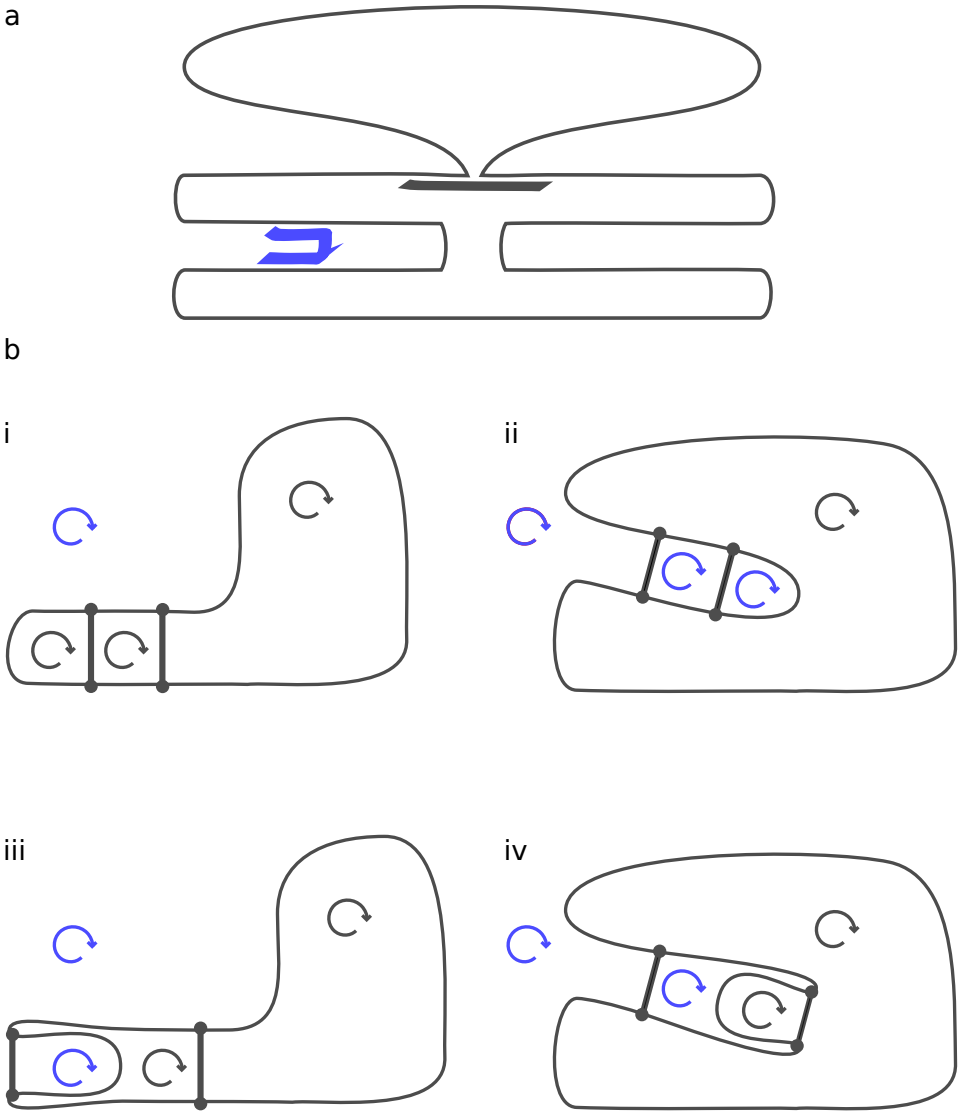


Figure 6.5: (a): A scaffold with two fully bound staples. (b): Ambiguity arises when determining a set of loops that reflects the entropic cost of folding the scaffold. To construct a graph based on (a), vertices and edges corresponding to staple crossovers are created. Four planar embeddings of this graph are illustrated in (i)-(iv). In these embeddings, faces are considered either internal (black arrows) or external (blue arrows) relative to the scaffold. A staple-free scaffold has two faces (one internal and one external), and adding any number of staples increases the number of faces by an equal amount. As demonstrated by (i)-(iv), multiple embeddings of the same graph are possible, each resulting in a different set of faces. In the model, each face is interpreted as a looping constraint. We wish to determine, given a set of staples, a single entropic cost, and it is not obvious which set of loops best reflects the change of entropy in the physical system. This ambiguity is resolved by specifying *a priori* which staples are external to the scaffold; if blue staples are internal and grey external, (b.iv) is the appropriate planar projection.

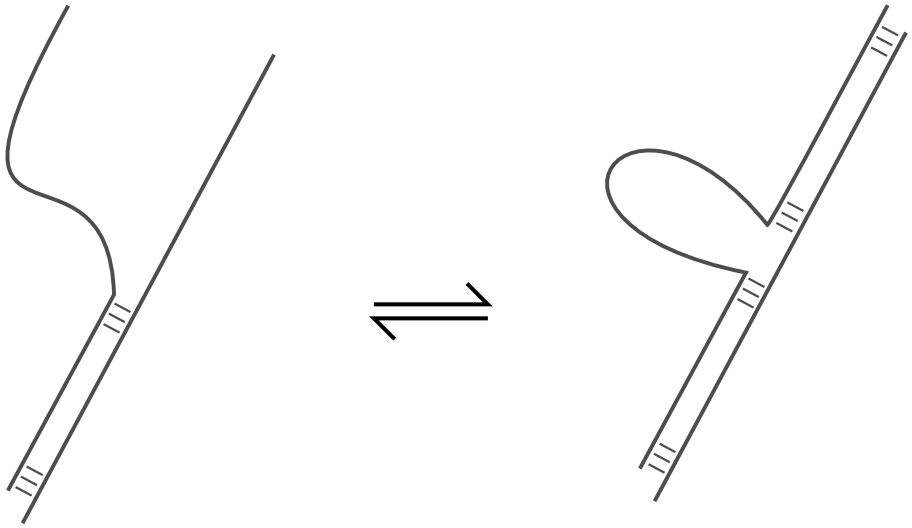


Figure 6.6: Bulge loop formation in a DNA duplex: two long duplex sections enclose a single-stranded loop. Bulge formation is a special case of loop formation within the context of DNA origami.

and composition of the loop.

To compute the effective concentration of the opposing domain, denoted as c_{eff}, we allow a small interaction radius r_c such that both domains need to be within distance r_c to initiate duplex formation. Let P_{r_c} be the probability that the two complementary domains are within distance r_c, then the effective rate is given by

$$\mathbf{R}(h, f) \approx k_+ c_{\text{eff}} \tag{6.21}$$

$$= k_+ \frac{P_{r_c}}{N_A v_{r_c}} \tag{6.22}$$

where $N_A = 6.022 \cdot 10^{23} \times \text{mol}^{-1}$ is Avogadro's number and v_{r_c} is the volume of a sphere with radius r_c. The DNA forming the loop is approximated as a freely-jointed chain consisting of two distinct segment types, being double- and single-stranded DNA. Let the end-to-end distance of the chain be given by R, denote by $P(\mathsf{R})$ its probability distribution, then $P_{r_c}^{\text{loop}} = \int_0^{r_c} P(\mathsf{R}) d\mathsf{R}$ is the probability that the ends of the loop are within r_c, resulting in

$$\mathbf{R}(h, f) \approx k_+ \frac{\int_0^{r_c} P(\mathsf{R}) d\mathsf{R}}{N_A v_{r_c}}. \tag{6.23}$$

For a chain with m distinct segment-types, for a large number of segments,

$$P(\mathsf{R}) = 4\pi \mathsf{R}^2 \left(\frac{3}{2\pi E[\mathsf{R}^2]} \right)^{3/2} \exp\left(\frac{-3\mathsf{R}^2}{2E[\mathsf{R}^2]} \right), \tag{6.24}$$

where $E[\mathsf{R}^2] = \sum_{i \leq m} N_i b_i^2$ is the mean-squared distance between the two ends. Here, N_i is the number of segments of type i with Kuhn length b_i. For $m = 1$ the expression is a classic result of statistical physics [161, 162] that we repeat here.

Freely-jointed chain

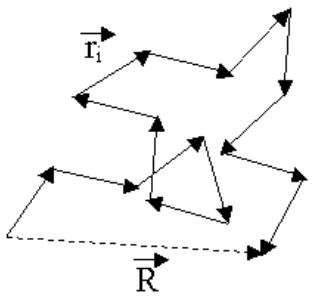


Figure 6.7: Freely-jointed chain. (public domain)

The freely-jointed chain is a statistical model of random behaviour of polymers. Let the

chain consist of N segments of length d and let each segment $\vec{r}_i \in \mathbb{R}^3$ have a uniformly random orientation, so that the probability distribution is given as

$$P(\vec{r}_i) = \frac{1}{4\pi d^2}\delta(|\vec{r}_i| - d). \tag{6.25}$$

The end-to-end distance R, the expected end-to-end distance $E[\mathsf{R}]$, and expected squared distance $E[\mathsf{R}^2]$ are given as

$$\mathsf{R} = \left|\sum_{i=0}^{N-1} \vec{r}_i\right| \qquad E[\mathsf{R}] = 0 \tag{6.26}$$

$$E[\mathsf{R}^2] = \sum_{i=0}^{N-1} |\vec{r}_i|^2 = Nd^2 \qquad \text{(by independence of } \vec{r}_i, \vec{r}_j \text{ when } i \neq j) \tag{6.27}$$

Let the components of R along some axes be $\mathsf{R_x}, \mathsf{R_y}, \mathsf{R_z}$, then

$$E[\mathsf{R_x}] = E[\mathsf{R_y}] = E[\mathsf{R_z}] = 0 \tag{6.28}$$

$$E[\mathsf{R}^2] = E[\mathsf{R_x}^2 + \mathsf{R_y}^2 + \mathsf{R_z}^2] = E[\mathsf{R_x}^2] + E[\mathsf{R_y}^2] + E[\mathsf{R_z}^2] \tag{6.29}$$

$$E[\mathsf{R_x}\mathsf{R_y}] = E[\mathsf{R_x}\mathsf{R_z}] = E[\mathsf{R_z}\mathsf{R_y}] = 0 \tag{6.30}$$

and by the arbitrary orientation of the axis we obtain $E[\mathsf{R_x}^2] = E[\mathsf{R_y}^2] = E[\mathsf{R_z}^2] = \frac{1}{3}E[\mathsf{R}^2]$. The probability distribution of the end-to-end distance R, obtained for large N by a normal approximation of the x, y, z components of R

$$f_x = \frac{1}{\sqrt{2\pi\frac{1}{3}E[\mathsf{R}^2]}} \exp \frac{-3\mathsf{R}_x^2}{2E[\mathsf{R}^2]} \tag{6.31}$$

$$P(\mathsf{R}) = \int_0^{2\pi}\int_0^{\pi} f_x f_y f_z \mathsf{R}^2 \sin\theta \, d\theta \, d\phi \tag{6.32}$$

$$= 4\pi\mathsf{R}^2 \left(\frac{3}{2\pi E[\mathsf{R}^2]}\right)^{3/2} \exp\left(\frac{-3\mathsf{R}^2}{2E[\mathsf{R}^2]}\right), \tag{6.33}$$

is a basic result in statistical physics [161, 162]. Eq. 6.33 is a good approximation to the exact distribution when $R < d\sqrt{N}$ (see [161]). To see that the result holds for $m > 1$ observe that Eq. 6.24 is the probability density for a multivariate Gaussian process. The combined end-to-end distance of two chains A and B with two distinct segment lengths is therefore given by another Gaussian with $\sigma^2 = E[\mathsf{R}_A^2] + E[\mathsf{R}_B^2]$, when we assume both chains to contain a large number of segments.

Substituting Eq. 6.24 into Eq. 6.23 we obtain

$$\mathbf{R}(h,f) \approx k_+ c_{\text{eff}} \tag{6.34}$$

$$= k_+ \frac{\int_0^{r_c} P(\mathsf{R}) d\mathsf{R}}{v_{r_c} N_A} \tag{6.35}$$

$$= k_+ \left(\frac{\int_0^{r_c} 4\pi \mathsf{R}^2 \left(\frac{3}{2\pi E[\mathsf{R}^2]_{\text{loop}}} \right)^{\frac{3}{2}} \exp\left(\frac{-3\mathsf{R}^2}{2E[\mathsf{R}^2]_{\text{loop}}} \right) d\mathsf{R}}{\frac{4}{3}\pi r_c^3 N_A} \right) \tag{6.36}$$

$$\approx k_+ \left(\frac{\int_0^{r_c} 4\pi \mathsf{R}^2 d\mathsf{R}}{\frac{4}{3}\pi r_c^3 N_A} \left(\frac{3}{2\pi E[\mathsf{R}^2]_{\text{loop}}} \right)^{3/2} \right) \tag{6.37}$$

$$= \frac{k_+}{N_A} \left(\frac{3}{2\pi E[\mathsf{R}^2]} \right)^{3/2} \tag{6.38}$$

in which we have assumed $r_c \ll E[\mathsf{R}^2]_{\text{loop}}$ so that $\exp\left(\frac{-3\mathsf{R}^2}{2E[\mathsf{R}^2]_{\text{loop}}} \right) \approx 1$ to approximate Eq. 6.36. Using Eq. 6.20 we write

$$\Delta G^{\text{loop}} = -RT\gamma \ln \frac{C_\gamma}{E[\mathsf{R}^2]} \tag{6.39}$$

with $\gamma = 3/2$ and $C_{3/2} = \left(\frac{1 \times \text{M}^{-1}}{N_A} \right)^{2/3} \frac{3}{2\pi}$. We express ΔG^{loop} in this way as it allows parameters that generalize our description. $\gamma = 3/2$ is the well-known loop-exponent of a freely-jointed chain [163]. It gives the scaling of the typical volume accessible to the end of a polymer with the polymer's contour length. Allowing an excluded volume around the chain tends to swell a polymer chain, so that an effective $\gamma > 3/2$ is obtained [163]. Theoretical estimates predict $\gamma \sim 1.75$ for a three-dimensional self-avoiding walk [163]. The widely-used SantaLucia model uses a value as high as $\gamma = 2.44$ in an equivalent calculation for a purely single-stranded bulge loop, and is fitted to measurements on DNA loop formation [26, 164].

The parameters γ and C affect ΔG^{loop} in different ways. Increasing γ at fixed C exaggerates the differences in ΔG^{loop} between longer and shorter loops, whilst also making all loops less stable. Increasing C at fixed γ makes all loops more stable by a uniform amount. We explore the properties of our model as γ and C are modulated. In particular, we consider the consequences of varying γ while modulating C so that ΔG^{loop} for an 18-base single-stranded 'bulge' loop is fixed at the value obtained in the freely-jointed case. γ and C are modulated together because changing γ at fixed C might not result in reasonable values of ΔG^{loop}. As an illustration, we plot ΔG^{loop} for a purely single-stranded loop as a function of length for $\gamma = 1.5$, 2.5 and 3.5 in Fig. 6.8. The values of $C_{2.5}, C_{3.5}$ are fitted to obtain the same value ΔG^{loop} as the freely-jointed chain model at length 18 nt, resulting in the values of Table 6.1. For comparison, we also show the equivalent quantity as estimated by the SantaLucia model, which uses $\gamma = 2.44$ for longer

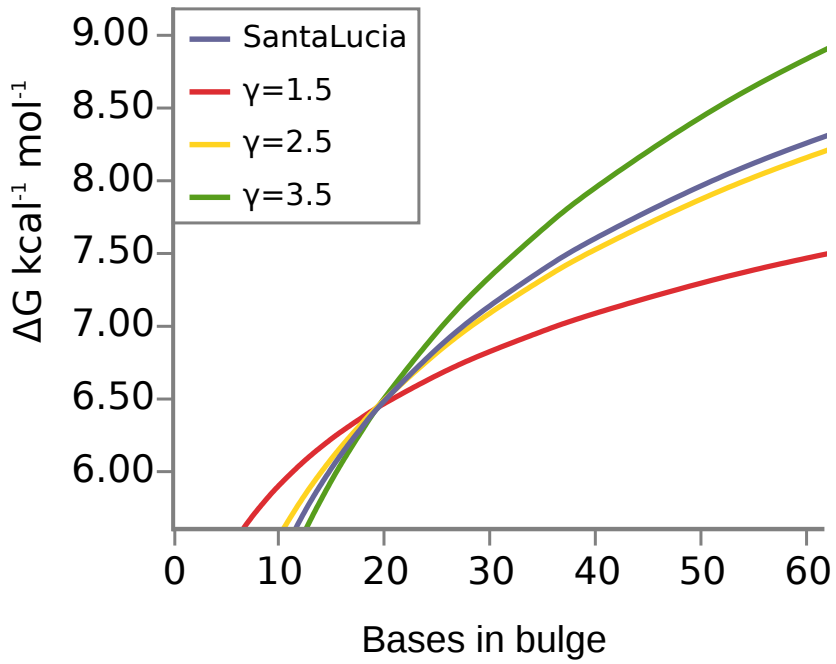


Figure 6.8: Comparison between the free energy costs of forming a single-stranded bulge loop for different values of the parameter γ at $T = 60\,^{\circ}\mathrm{C}$. The comparable quantity from SantaLucia's nearest-neighbor model [26] is also plotted for comparison.

γ	C_γ
1.5	6.7×10^{-19} m^2
2.5	2.8×10^{-18} m^2
3.5	5.2×10^{-18} m^2

Table 6.1: Model parameters: volume exclusion exponent γ and corresponding C_γ for ΔG^{loop}. $C_{2.5}, C_{3.5}$ are fitted to obtain ΔG^{loop} equal to that of $\gamma = 1.5$ for a loop of length 18 nt.

loops [26].

Given a loop with a certain content of duplex, single-stranded and staple crossovers, $E[\mathsf{R}^2] = \sum_{i \le m} N_i b_i^2$ is estimated as follows. A double-stranded domain is treated as a single segment of length $\lambda_{x,ds}$, equal to the length of the helix, calculated using a contour length of 0.34 nm per base for dual-stranded DNA, so that $\lambda_{x,ds} = 0.34 \times x$ nm for a domain with x base-pairs [165]. We treat a single-stranded domain as consisting of L/λ_{ss} segments of length λ_{ss}, where $\lambda_{ss} = 1.8$ nm is approximately the Kuhn length of single-stranded DNA [166], and L is the contour length of the single-stranded section. In our model we use a contour length of 0.6 nm per base for single stranded DNA, so that $L = 0.6 \times x$ nm for a domain with x number of bases [166]. Thus each single-stranded domain of x bases contributes $x/3$ Kuhn lengths of ssDNA to the loop. Where two scaffold domains are held together by a staple, we represent the link by a segment of length λ_{ss}. For a chain of N_x segments of length $\lambda_{x,ds}$ and M segments of length λ_{ss}, we find $E[\mathsf{R}^2] = \sum_x N_x \lambda_{x,ds}^2 + M\lambda_{ss}^2$.

6.5 Simulation method

The state space S, rate matrix $\mathbf{R}_{T(t)}$ and initial condition form an inhomogeneous continuous-time Markov chain $(S, \mathbf{R}_{T(t)}, \pi_0)$ (Def. 3.1.13), where s_0 is the state indicating the empty scaffold and $T(t)$ is the externally imposed function relating time and temperature that cycles between the initial temperature T_{start}, the ending temperature T_{end} and then back to the initial temperature with constant $|\mathrm{d}T/dt|$. Instead of simulating the inhomogeneous CTMC directly, we approximate $T(t)$ and hence the rate matrix $\mathbf{R}_{T(t)}$ as piecewise constant across 1 second intervals, which is a reasonable approximation for typical cooling rates in experiments. Individual traces are then generated by applying the standard Gillespie simulation algorithm at each interval, as in Def. 3.1.12. To implement the Gillespie algorithm, it is necessary to calculate all rates $\mathbf{R}(s, s')$ from the current state s to alternative states s'. $\mathbf{R}(s, s')$ will be non-zero in the following cases (see Fig. 6.9):

1 All unbound domains can hybridize to complementary domains on staples, with a rate given by Eq. 6.3.
2 Domains of half-bound staples can unbind, with a rate given by Eq. 6.4.
3 Domains of fully-bound staples can unbind, with a rate given by Eq. 6.6.
4 Half-bound staples can become fully-bound if the opposing domain is free, with a rate given by Eq. 6.7.

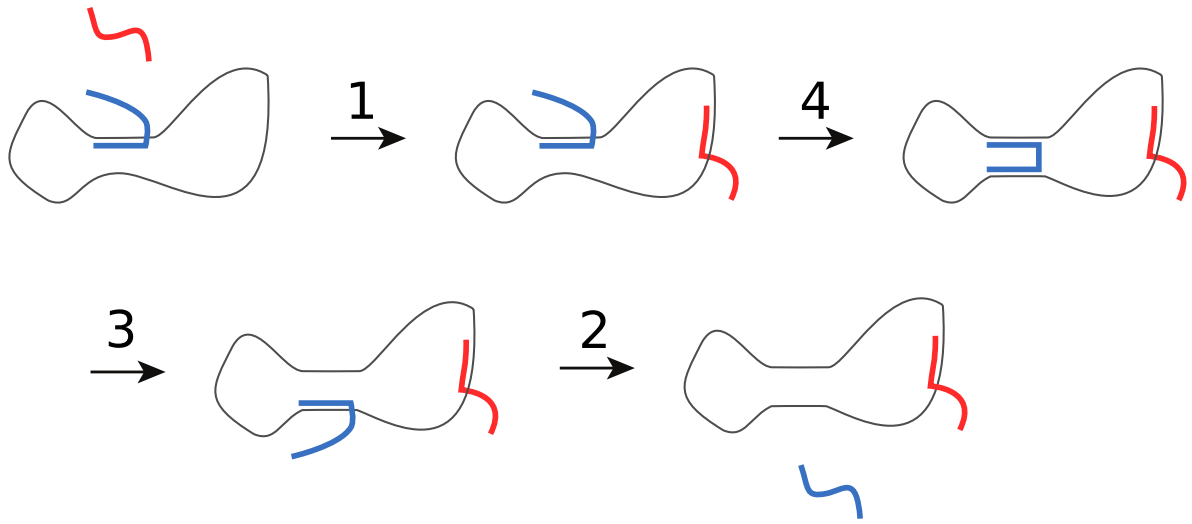


Figure 6.9: A simulation of a DNA origami self-assembly, where each transition is labelled with the corresponding transition type. In the simulation the red staple binds to the scaffold (1), the half-bound blue staple becomes fully bound (4) and then again half-bound (3), and finally releases from the scaffold (2). Approximately 400,000 transitions occur in single simulation of annealing and subsequent melting of the tile in Fig. 6.1, when using $dT/dt = 1.0\,^{\circ}\mathrm{C\ min}^{-1}$.

The term ΔG^{duplex}, required in transition types 2 and 3, is computed straightforwardly using

the model of SantaLucia et al. [26] and depends on the nucleotide sequence of the domain. The term ΔG^{stack} is also needed in transition types 2 and 3, and is computed using a simple lookup of the status of neighbouring domains. Finally, the term ΔG^{shape} occurs in transition types 2 and 4, and this is where the global and local model differ. In both the global and local model, graphs representing state s and s' are needed to calculate ΔG^{shape}. As the simulation moves from state to state, the graph is updated. In the global model, a specific planar embedding of the graph representing state s, as depicted in Fig. 6.5, is used. This specific embedding is not required in the local model.

The graph $\mathsf{H}(s) = (\mathsf{V}, \mathsf{E}(s))$ itself is defined as follows: each join between domains on the scaffold is a vertex $v \in \mathsf{V}$ and each domain is an edge $e \in \mathsf{E}(s)$ between the appropriate vertices. Fully-bound staples also define an additional edge between the two vertices that are linked by the staple, and an example of such a graph is given in Fig 6.10. A labelling function

$$\mathsf{L} : \mathsf{E} \to \{\text{single-stranded}, \text{double-stranded}, \text{crossover}\} \tag{6.40}$$

assigns the status of each edge, which also has a fixed length (number of nucleotides) if it is a scaffold domain rather than a crossover. Each edge $e \in \mathsf{E}(s)$ is weighted as follows:

$$\mathsf{W}(e) = \begin{cases} \lambda^2_{x,ds} & \text{if } \mathsf{L}(e) = \text{double-stranded of length } x, \\ \dfrac{x}{3}\lambda^2_{ss} & \text{if } \mathsf{L}(e) = \text{single-stranded of length } x, \\ \lambda^2_{ss} & \text{if } \mathsf{L}(e) = \text{crossover}. \end{cases} \tag{6.41}$$

The total weight of any loop (simple cycle) within the graph is then $E[\mathsf{R}^2]$, the key quantity in estimating the loop cost (Eq. 6.39). In the global model the graph $\mathsf{H}(s)$ is assumed to have an unique planar embedding, identified by a set of faces. Faces are defined in the regular sense, i.e., they are subgraphs of $\mathsf{H}(s)$

$$F(s) = \{(\mathsf{V}_i, \mathsf{E}_i) | \mathsf{V}_i \subseteq \mathsf{V}, \mathsf{E}_i \subseteq \mathsf{E}(s)\} \tag{6.42}$$

where each set of edges E_i forms a cycle around a region in the embedding. The weight of each face $\mathsf{F}_i \in F(s)$ is given by the weight of the edges.

$$\mathsf{W}(\mathsf{F}_i) = \sum_{e \in \mathsf{E}_i} \mathsf{W}(e). \tag{6.43}$$

As the simulation progresses, the faces of the graph are merged (transition type 3) or split (transition type 4), and we use a custom data structure to dynamically update the faces of the graph. Each face represents a looping constraint, and $E[\mathsf{R}^2]$ for the loop is given by summing

the weights of the edges that surround the face. The shape term in the model is found as (cf. Eq. 6.13)

$$\Delta G_s^{\text{shape}} = \sum \Delta G^{\text{loop}} \tag{6.44}$$

$$= - \sum_{\mathsf{F}_i \in F(s)} RT\gamma \ln \frac{C_\gamma}{\mathsf{W}(\mathsf{F}_i)} \tag{6.45}$$

During a transition, it is only necessary to recalculate $E[\mathsf{R}^2]$ and ΔG^{loop} for the affected loops.

The local model does not require an embedded graph representation, but uses the same weighted graph $\mathsf{H}(s)$ representing s. Estimating the change of ΔG^{shape} is only necessary for transitions where half-bound staples become fully bound (transition type 4), and in that case we approximate

$$\Delta G^{\text{shape}} = \Delta G^{\text{loop}}_{\text{min}}, \tag{6.46}$$

where $\Delta G^{\text{loop}}_{\text{min}}$ is the minimal ΔG^{loop} for a loop incorporating the newly formed staple crossover in the new state s'. This corresponds to finding a simple cycle, which has to include the to-be-formed crossover, that minimizes $E[\mathsf{R}^2]$. We employ Dijkstra's shortest path algorithm [167] to identify this loop by searching for the shortest path in s' between the two vertices that are to be connected by the staple crossover. Given a state s and a half-bound staple p, let v_1, v_2 be the vertices that are joined by a new edge once p becomes fully bound by hybridization to domain e. Let the new graph H' be equal to $\mathsf{H}(s)$ except that $\mathsf{L}(e) =$ double-stranded, and let $\mathsf{DIJKSTRA}(v_1, v_2)$ be the weight of the shortest path between v_1, v_2 in H' under W. Then

$$\Delta G^{\text{shape}}_{\text{min}} = -RT\gamma \ln \frac{C}{E[\mathsf{R}^2]_{\text{min}}} \tag{6.47}$$

$$E[\mathsf{R}^2]_{\text{min}} = \lambda_{ss}^2 + \mathsf{DIJKSTRA}(v_1, v_2). \tag{6.48}$$

In the next section we give examples of rate calculations for each move type.

6.6 Example rate calculations

In Fig. 6.10 a partially folded origami is shown in states $\mathsf{a} - \mathsf{d}$, and we discuss how the term ΔG^{shape} is computed for the transitions $\mathbf{R}(\mathsf{a}, \mathsf{b}), \mathbf{R}(\mathsf{a}, \mathsf{c}), \mathbf{R}(\mathsf{b}, \mathsf{a}), \mathbf{R}(\mathsf{c}, \mathsf{a})$ and $\mathbf{R}(\mathsf{d}, \mathsf{a})$, which form an illustrative set of rate calculations. Firstly, in both the global and local models, initial binding from solution and unbinding of a second staple domain do not require estimation of ΔG^{shape}.

Transitions $\mathbf{R}(\mathsf{b},\mathsf{a})$ and $\mathbf{R}(\mathsf{c},\mathsf{a})$ fall into these categories. Following Eq. 6.3 and Eq. 6.6, we find

$$\mathbf{R}(\mathsf{c},\mathsf{a}) = k_+[p] \quad \text{and} \tag{6.49}$$

$$\mathbf{R}(\mathsf{b},\mathsf{a}) = k_+ \exp\left(\Delta G^{\text{duplex}}_{\mathsf{a},\mathsf{b}}/RT\right) \times \mathrm{M}, \tag{6.50}$$

in which $\Delta G^{\text{duplex}}_{\mathsf{a},\mathsf{b}}$ is the estimated standard free-energy change of formation of the duplex in question and $[p]$ is the staple concentration. Note that there are no domain stacking terms in this example.

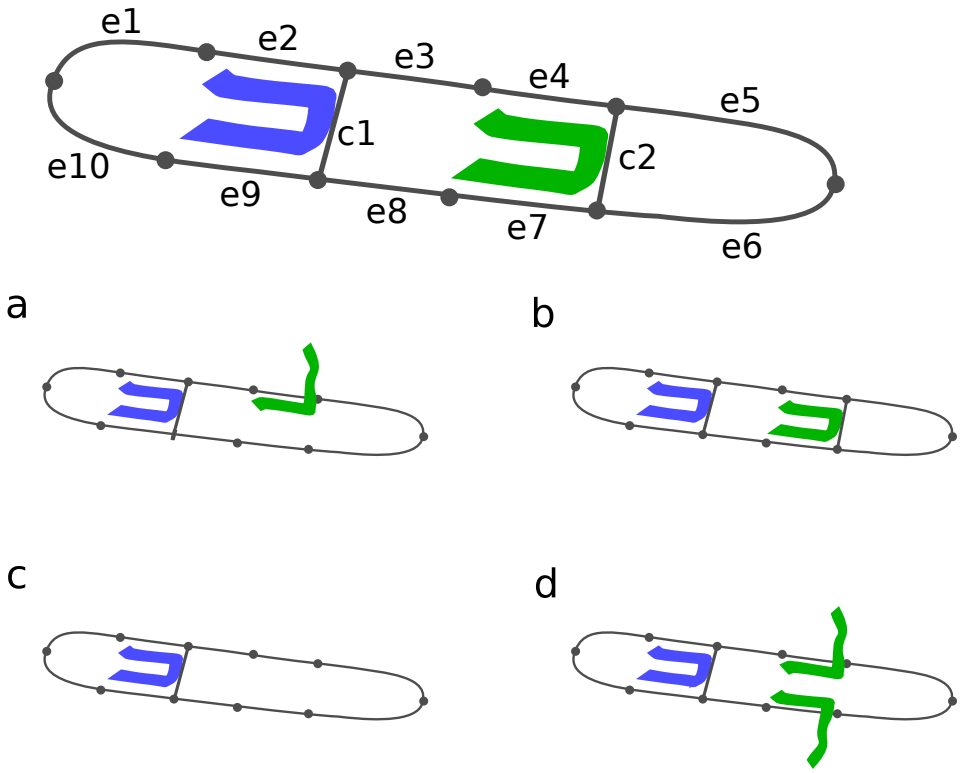


Figure 6.10: Graph-representation with labelled edges of a partially folded origami in four $(\mathsf{a}-\mathsf{b})$ states. Top: The internal graph representation with named edges. Edges named 'e' represent single or double-stranded domains and 'c' edges represent crossover connections mediated by fully-bound staples. a) The origami with one fully bound and one half-bound staple. b) The origami with two fully bound staples. c) The origami with one fully bound staple. d) The origami with one fully bound and two half-bound staples.

The other transitions require estimates of ΔG^{shape}, at least for the global model. In Table 6.2, we identify the faces and associated loop costs which will be of relevance in the global model (a subset of loop costs are also relevant to the local model). We remind the reader that loop costs are determined by $E[\mathsf{R}^2]$, which is in turn given by summing over the weights of individual

State	Face	Edges	Weight	$E[\mathsf{R}^2]$ /nm^2
s_a	F_1	$e_1, \underline{e}_2, c_1, \underline{e}_9, e_{10}$	$2 \times \lambda^2_{x,ds} + (2x/3+1) \times \lambda^2_{ss}$	97.0
	F_2	$e_3, \underline{e}_4, e_5, e_6, e_7, e_8, c_1$	$1 \times \lambda^2_{x,ds} + (5x/3+1) \times \lambda^2_{ss}$	119.2
	F_3	$e_1, \underline{e}_2, e_3, \underline{e}_4, e_5, e_6, e_7, e_8, \underline{e}_9, e_{10}$	$3 \times \lambda^2_{x,ds} + (7x/3+0) \times \lambda^2_{ss}$	209.7
s_b	F_1	$e_1, \underline{e}_2, c_1, \underline{e}_9, e_{10}$	$2 \times \lambda^2_{x,ds} + (2x/3+1) \times \lambda^2_{ss}$	97.0
	F_2	$e_3, \underline{e}_4, c_2, \underline{e}_7, e_8, c_1$	$2 \times \lambda^2_{x,ds} + (2x/3+2) \times \lambda^2_{ss}$	100.2
	F_3	e_5, e_6, c_2	$0 \times \lambda^2_{x,ds} + (2x/3+1) \times \lambda^2_{ss}$	37.8
	F_4	$e_1, \underline{e}_2, e_3, \underline{e}_4, e_5, e_6, \underline{e}_7, e_8, \underline{e}_9, e_{10}$	$4 \times \lambda^2_{x,ds} + (6x/3+0) \times \lambda^2_{ss}$	222.0
s_c	F_1	$e_1, \underline{e}_2, c_1, \underline{e}_9, e_{10}$	$2 \times \lambda^2_{x,ds} + (2x/3+1) \times \lambda^2_{ss}$	97.0
	F_2	$e_3, e_4, e_5, e_6, e_7, e_8, c_1$	$0 \times \lambda^2_{x,ds} + (6x/3+1) \times \lambda^2_{ss}$	106.9
	F_3	$e_1, \underline{e}_2, e_3, e_4, e_5, e_6, e_7, e_8, \underline{e}_9, e_{10}$	$2 \times \lambda^2_{x,ds} + (8x/3+0) \times \lambda^2_{ss}$	197.3
s_d	F_1	$e_1, \underline{e}_2, c_1, \underline{e}_9, e_{10}$	$2 \times \lambda^2_{x,ds} + (2x/3+1) \times \lambda^2_{ss}$	97.0
	F_2	$e_3, \underline{e}_4, e_5, e_6, \underline{e}_7, e_8, c_1$	$2 \times \lambda^2_{x,ds} + (4x/3+1) \times \lambda^2_{ss}$	131.5
	F_3	$e_1, \underline{e}_2, e_3, \underline{e}_4, e_5, e_6, \underline{e}_7, e_8, \underline{e}_9, e_{10}$	$4 \times \lambda^2_{x,ds} + (6x/3+0) \times \lambda^2_{ss}$	222.0

Table 6.2: Weights for the faces in the global model. For simplicity, all scaffold domains have an intrinsic length of $x = 16\,\text{nt}$. Underline indicates the edge is double-stranded.

edges in a loop. These edges contribute (Eq. 6.41):

$$\mathsf{W}(c_i) = \lambda_{ss} = (1.8\,\text{nm})^2 \text{ for a crossover;} \tag{6.51}$$

$$\mathsf{W}(e_i) = \begin{cases} \lambda^2_{x,ds} = (0.34\,\text{nm} \times x)^2 & \text{for double-stranded DNA with } x \text{ nt} \\ \dfrac{x}{3}\lambda^2_{ss} = (1.8\,\text{nm})^2 \times \dfrac{x}{3} & \text{for single-stranded DNA with } x \text{ nt} \end{cases} \tag{6.52}$$

where in our example $x = 16$ nt for all domains. The above expressions are used in Table 6.2 to calculate $E[\mathsf{R}^2]$ and ΔG^{loop} for the various faces in the global model.

Global model

In the global model the shape term is computed as

$$\Delta G^{\text{shape}}_{s,s'} = \sum_{\mathsf{F}_i \in F(s')} \Delta G^{\text{loop}}(\mathsf{W}(\mathsf{F}_i)) - \sum_{\mathsf{F}_i \in F(s)} \Delta G^{\text{loop}}(\mathsf{W}(\mathsf{F}_i)) \tag{6.53}$$

where $\mathsf{W}(\mathsf{F}_i)$ is the weight of a face in the embedded graph of state s. For $\gamma = 2.5$ we find at $T = 37.0\,^\circ\text{C}$

$$\Delta G^{\text{shape}}_{\mathsf{a},\mathsf{b}} = -3.828 \text{ kcal/mol,} \tag{6.54}$$

$$\Delta G^{\text{shape}}_{\mathsf{a},\mathsf{c}} = 0.261 \text{ kcal/mol, and} \tag{6.55}$$

$$\Delta G^{\text{shape}}_{\mathsf{d},\mathsf{a}} = 0.239 \text{ kcal/mol.} \tag{6.56}$$

using Eq. 6.39. Combining these specific values with Eq. 6.4 and Eq. 6.7. we find

$$\mathbf{R}(\mathsf{a},\mathsf{c}) = 0.6546 \cdot k_+ \exp\left(\Delta G^{\text{duplex}}_{\mathsf{c},\mathsf{a}}/RT\right) \times \text{M}, \tag{6.57}$$

$$\mathbf{R}(\mathsf{d},\mathsf{a}) = 0.6782 \cdot k_+ \exp\left(\Delta G^{\text{duplex}}_{\mathsf{a},\mathsf{d}}/RT\right) \times \text{M}, \tag{6.58}$$

$$\mathbf{R}(\mathsf{a},\mathsf{b}) = 1.998 \cdot 10^{-3} \times k_+ \times \text{M}. \tag{6.59}$$

Local model

For transitions $\mathbf{R}(\mathsf{a},\mathsf{c})$ and $\mathbf{R}(\mathsf{d},\mathsf{a})$, the local model uses $\Delta G^{\text{shape}} = 0$, as no loops are actually formed during these transitions. Thus

$$\mathbf{R}(\mathsf{a},\mathsf{c}) = k_+ \exp\left(\Delta G^{\text{duplex}}_{\mathsf{c},\mathsf{a}}/RT\right) \times \text{M}, \tag{6.60}$$

$$\mathbf{R}(\mathsf{d},\mathsf{a}) = k_+ \exp\left(\Delta G^{\text{duplex}}_{\mathsf{a},\mathsf{d}}/RT\right) \times \text{M}. \tag{6.61}$$

In case of the transition $\mathbf{R}_{\mathsf{a},\mathsf{b}}$, a new loop is formed. We must find the cycle in b containing c_2 that minimizes $E[\mathsf{R}^2]$ – this is the loop consisting of edges

$$e_5, e_6, c_2 \tag{6.62}$$

$E[\mathsf{R}^2]$ for this loop has already been calculated in Table 6.2 for the purposes of the global model. Using this result along with Eq. 6.7, Eq. 6.15 and Eq. 6.39, we find

$$\mathbf{R}(\mathsf{a},\mathsf{b}) = 1.494 \cdot 10^{-3} \times k_+ \times \text{M} \tag{6.63}$$

6.7 Results and discussion

To simulate the assembly process we apply the model to a prototypical origami tile, depicted in Fig. 6.1. The tile of Fig. 6.1 is one half of a special polymorphic tile that we discuss in Chapter 7.

We compare the model at various settings, taking into account the following measures:

- Annealing temperature (T_a), the first point during the annealing stage at which a moving average of the fraction of hybridized domains surpasses 50%.
- Hysteresis, equal to $T_m - T_a$, where the melting temperature T_m is the first point at which a moving average of the fraction of hybridized domains is lower than 50% during the melting stage.
- Transition width ΔT_a, the difference in temperature between when 20% of the domains are hybridized to when 80% of domains are hybridized.

Each property is averaged over 160 independent simulations, which are generated using the Monte Carlo algorithm outlined in Section 6.5. The moving average is taken over 12 seconds. Unless stated otherwise the temperature drops by $1\,^{\circ}\mathrm{C\ min}^{-1}$ starting at $85\,^{\circ}\mathrm{C}$ until $25\,^{\circ}\mathrm{C}$, at which point the temperature is cycled back to $85\,^{\circ}\mathrm{C}$ at the same rate. We assume each staple to be present at a concentration of 20 nM and we assume the scaffold strands are dilute enough that reduction of staple concentration during folding can be neglected. Plots of average anneal/melting curves of the origami correspond to the average occupancy of all domains. These plots also include the anneal/melting curve for a single individual staple, which serves as contrast between the average and individual staple behaviour.

In addition to the model described thus far, we also consider the following modifications:

- The volume-exclusion parameter γ is varied between 1.5, 2.5 and 3.5, for which we apply the corresponding C_γ of Table 6.1.
- The stacking parameter is set to $\Delta G^{\mathrm{stack}} = n\langle \Delta G^{\mathrm{bp}}(T)\rangle)$, where $\langle \Delta G^{\mathrm{bp}}(T)\rangle$ is the average contribution of a base-pair and $n = 0, 1, 2, 3$.
- The domain-specific (sequence-specific) stabilities are modified to be equal to the average stability of a 16 base pair domain.
- The temperature gradient is varied between $0.1, 1.0$ and $10.0\,^{\circ}\mathrm{C\ min}^{-1}$.

In addition we present experimentally observed melting and annealing curves for the origami tile in Fig. 6.12, where 20 nM staple concentration, 10 nM scaffold concentrations and equal buffer conditions were used. Because our model assumes a constant staple concentration, in comparison to the experiment we expect our model to over-estimate the annealing temperature T_a and hence underestimate the hysteresis ($\Delta T_a = T_m - T_a$) (typically, a higher ratio of staple excess is used, in which case the assumption of constant staple concentration is more reasonable).

6.7.1 Basic behaviour

We first display the behaviour of the global model using $\gamma = 1.5$ and without stacking stabilization at nicks, that is, $n = 0$ in $\Delta G^{\mathrm{stack}}(T) = n\langle \Delta G^{\mathrm{bp}}(T)\rangle$. The average fraction of domains that are bound as a function of temperature during both annealing and melting is shown in Fig. 6.11a. In addition, we show the degree of incorporation of a single typical staple (staple X – highlighted in Fig. 6.11a). The first observation from Fig. 6.11a is that origami assembly occurs at $T_a \approx 65\,^{\circ}\mathrm{C}$. This midpoint of the melting transition is consistent with that observed in the equivalent experimental system (Fig. 6.12).

A second observation is that the formation transition in the model shows little hysteresis; annealing and melting curves nearly overlap, despite the rapid rate of cooling. Hysteresis is a generic feature of origami systems [153, 154], and significant hysteresis is also observed in our

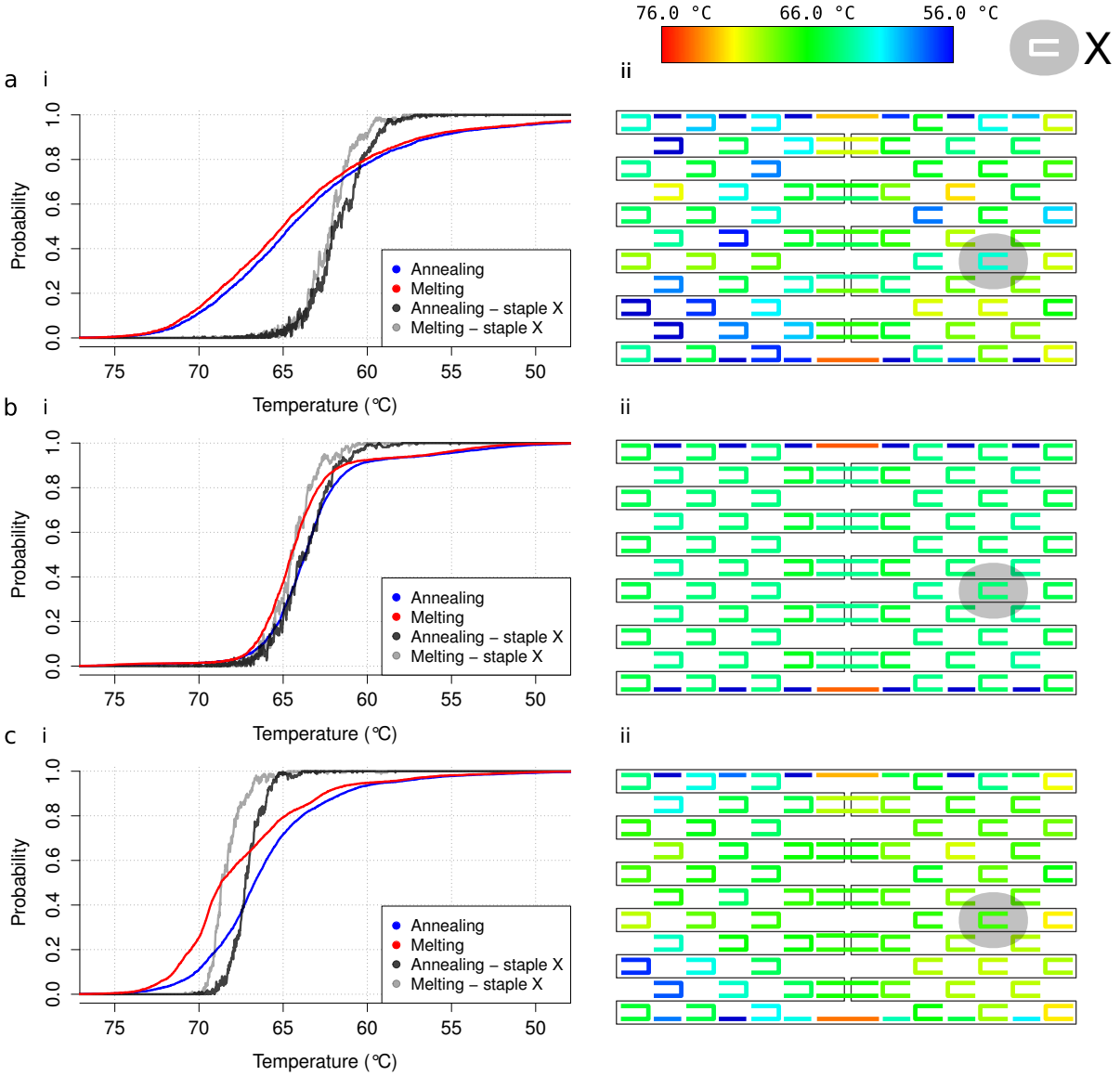


Figure 6.11: Staple binding as a function of temperature during cooling and heating for a simple origami using the global model. ai) average number of scaffold domains that are occupied as a function of temperature. Smoothed curves are plotted using $\gamma = 1.5$ and no contribution from stacking at nicks. Also shown is the probability of two-domain binding for a specific staple X, the highlighted staple. aii) indicates the annealing temperature of each staple separately. Staples with an incorporation temperature $< 56\,^{\circ}\mathrm{C}$ are plotted dark blue. b) shows identical data for a system in which each domain stability is set to equal the average for a 16 base-pair domain. c) shows equivalent data to ai and aii, but obtained using stacking strength $n = 2$ and $\gamma = 2.5$.

experimental measurements (Fig. 6.12). It is clear that the basic model with $\gamma = 1.5$, and no stabilizing contribution of the stacking at nicks, fails to capture this effect.

Thirdly, the melting transition is fairly broad; the transition from 20% of domains bound to 80% of domains bound during annealing occurs over a temperature range of approximately $\Delta T_a \approx 9°\mathrm{C}$. Transition widths observed in experimental data in Fig. 6.12a are sharper than

T_a/ °C

	Global model - γ			Local model - γ		
Stacking n	1.5	2.5	3.5	1.5	2.5	3.5
0	64.6	63.4	61.9	64.6	63.2	61.7
1.0	66.6	65.2	63.8	66.4	65.1	63.8
2.0	68.0	66.7	65.5	67.9	66.6	65.5
3.0	69.1	67.9	66.7	69.0	67.8	66.7

Hysteresis/ °C

0	0.1	0.4	0.8	0.2	0.4	0.6
1.0	0.6	1.0	1.3	0.7	0.8	1.0
2.0	1.5	1.8	1.9	1.4	1.7	1.7
3.0	2.2	2.5	2.7	2.3	2.5	2.4

ΔT_a/ °C

0	8.7	8.6	8.6	8.8	8.8	9.1
1.0	6.6	6.5	6.5	6.7	6.7	6.8
2.0	5.2	5.1	5.0	5.4	5.3	5.2
3.0	4.4	4.2	4.3	4.4	4.3	4.3

Table 6.3: Model properties as a function of exponent γ and stacking strength n (base pair equivalents, $\Delta G^{\text{stack}}(T) = n\langle\Delta G^{\text{bp}}(T)\rangle$). Data reported for annealing temperature T_a hysteresis and transition width ΔT_a. Data is presented for heating/cooling rates of 1 °C min^{-1} for the origami shown in Fig. 6.1. Standard errors of the mean, estimated from 160 independent cycles, are smaller than 0.1 °C for all data reported here. Results are reported for both global and local models.

those predicted by the model (Fig. 6.11a). The wide annealing transition in the model is not due to the individual staples having broad transitions. For example, staple X goes from 20% to 80% bound over a temperature range of $\Delta T_a^X \approx 2.5\,°\text{C}$. This width is representative of the annealing behaviour of other staples; variation in staple-specific T_m occurs as some sequences are more stable and due to differences in enclosed loops. In Fig. 6.11b, we consider a system in which all domains, except the two longer domains of 32 base pairs, are assigned a domain stability $\Delta G^{0\,\text{duplex}}$ equal to that of a 16-bp domain, averaged over all possible sequences. The transition width ΔT_a is drastically reduced, and is nearly equal to that of the individual staple X.

6.7.2 Exploring the parameter space

We now explore the dependence of behaviour on model parameters. The key variable quantities in the model are loop parameters γ and C (see Eq. 6.39), and the stabilizing contribution of stacking at nicks $\Delta G^{\text{stack}}(T) = n\langle\Delta G^{\text{bp}}(T)\rangle$. Table 6.3 shows the variation in the annealing temperature hysteresis, and annealing width with γ and n. C_γ is varied with γ to ensure a

constant cost for an 18-base bulge, as discussed in Section 6.4.

The following trends are clear.

1. An increased contribution from stacking at nicks results in a higher melting annealing temperature, sharper transitions and increased hysteresis.
2. Increased γ leads to lower annealing temperatures and increased hysteresis, but has a weak effect on transition widths.

Fig. 6.11c illustrates these effects for $\gamma = 2.5$ and $n = 2$. The consequences of increased stabilization from nicks are explained as follows. Firstly, stacking at nicks is a net stabilizing effect, and it is therefore unsurprising that increases this contribution results in a higher annealing temperature T_a. Secondly, it is a cooperative interaction, meaning that the binding of one staple favours the subsequent binding of another. Cooperativity results in narrower folding transitions, because the binding of isolated staples is suppressed relative to the formation of well-formed regions, which exaggerates hysteresis.

The influence of γ is more subtle. The net effect of increasing γ, with C_γ adjusted to maintain the penalty for an 18-base single-stranded loop, is to increase ΔG^{loop} for longer loops. For the origami studied here, the majority of loops are longer than 18 nucleotides. As larger γ results in a larger penalty, we expect stronger cooperative effects for large γ, when the presence of other staples can substantially reduce loop costs for incoming staples. Consistent with this hypothesis, we see that increased γ leads to larger hysteresis (Table 6.3). Transition width ΔT_a, however, shows only a weak dependence on γ.

We investigate the phenomenon of hysteresis using the model. We employ a temperature ramp of $|\frac{dT}{dt}| = 10.0\,^\circ\text{C min}^{-1}$ and $|\frac{dT}{dt}| = 0.1\,^\circ\text{C min}^{-1}$ and plot the results in Fig. 6.13ab. Comparing the two protocols, we see that, on rapid heating, the melting temperature T_m increases by around $1.3\,^\circ$C, whereas the annealing temperature T_a decreases by $5.3\,^\circ$C. Similar behaviour was observed experimentally by Sobczak et al. [153], leading those authors to conclude that "folding rather than unfolding was not in equilibrium".

Variations of the model that were not explored

So far, we observed the effects of modifying the loop-exponent, the stabilizing contribution of nick stacking, and the temperature ramp. There are many other ways in which the model can be changed. For example:

- The annealing protocol could be adjusted to include an incubation period at constant temperature.

- The concentration of staples could be adjusted, which affects the expected time between binding events of the first type (see Fig. 6.9).
- The stiffness of single-stranded DNA, given by Kuhn length λ_{ss}, could be adjusted.
- The staples could be made to have more than two domains.
- k_+, the bi-molecular rate constant, set equal to 10^6 mol^{-1} s^{-1}, could be varied. Because k_+ acts as an effective scaling of time in the model, this is equivalent to changing the temperature ramp $\frac{dT}{dt}$.
- The secondary structure of the scaffold could be included in the model.
- The nucleotide sequence of the domains could be shifted.
- Partial hybridization of domain sequences, including nucleotide mismatches, could be included.
- Other origami designs could be simulated.

6.7.3 Global and local models

We now compare the local model with the global model that was used so far. The local model is a simpler but less rigorous alternative, and will prove useful in simulating the polymorphic origami tile in Chapter 7. To use the local model with confidence, however, it is important to establish how it differs from the global approach. Table 6.3 directly compares the local and global models in terms of annealing temperature, hysteresis and transition width. The predictions of the local and global models are largely similar, although the local model predicts a weaker dependence of hysteresis on γ. Because the (un)binding of one staple does not immediately (de)stabilize the staples that are already bound to the scaffold, the reduced dependency on γ is not surprising.

We set $\gamma = 2.5$ and $n = 2$ as parameter values which provide a reasonable match with the experimental data of Fig. 6.12, and compare the two approaches. In Fig. 6.14 a comparison between the two models is given. The averaged response is largely equal between model and experiment, and are consistent with the predictions of Table 6.3.

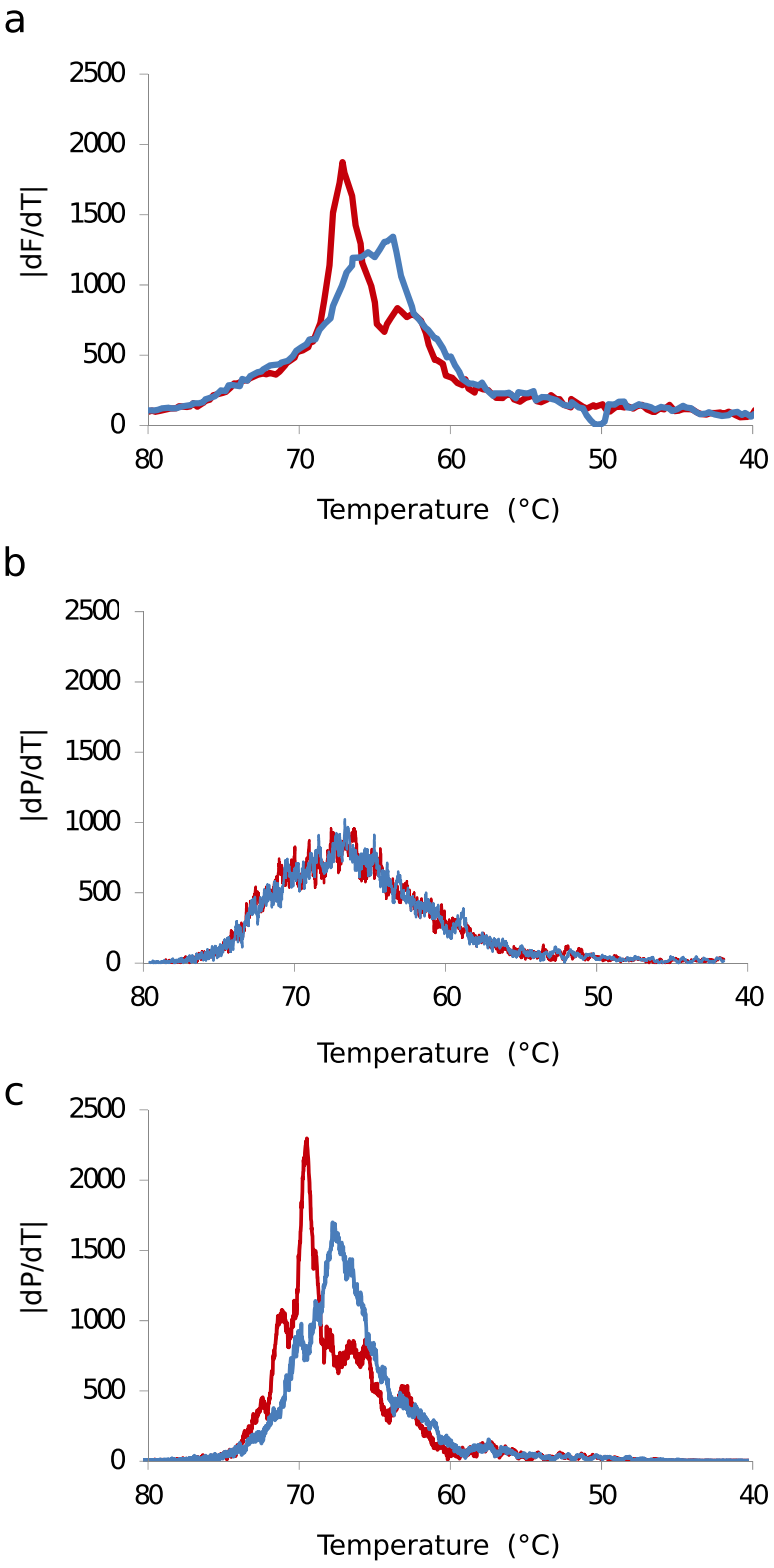


Figure 6.12: Fluorescence measurements obtained during the annealing of origami are compared to the model. SYBR® Green, a staining dye, preferentially binds to double stranded DNA and an increase in fluorescent response indicates an increase in the amount of double stranded DNA. Measurements are due to Dr. Jonathan Bath. a) dFluorescence/dt in arbitrary units is reported against temperature during cooling (blue) and heating (red) at a rate of 1 °C min^{-1}, for a system of 10 nM scaffold, 20 nM of each staple and 1× SYBR® Green. b) Derivative of the probability for a domain to be double stranded, as reported in the simulation using $\gamma = 1.5$ and $n = 0$ and scaled arbitrarily in the y-axis and smoothed over a $23s$ interval. c) as (b) but using $\gamma = 2.5$ and $n = 2$ and equal scaling.

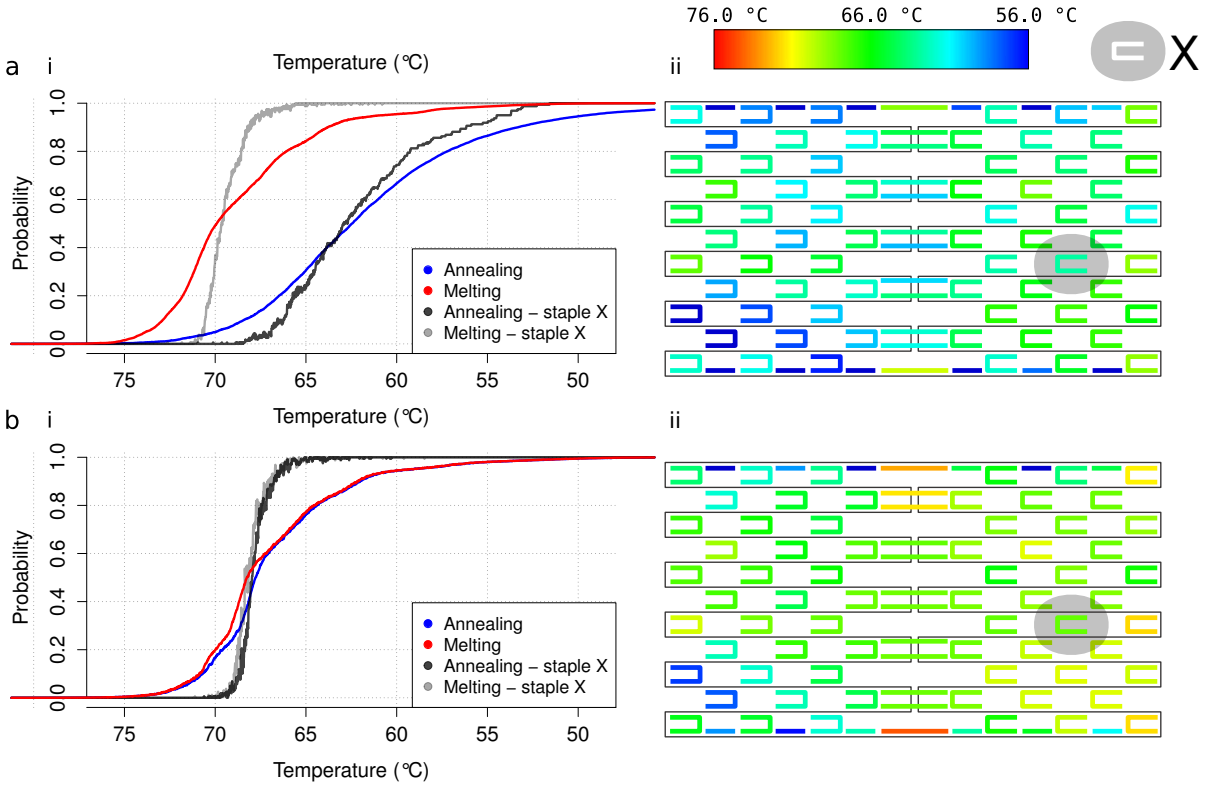


Figure 6.13: Melting and annealing curves for $n = 2$ and $\gamma = 2.5$ and a cooling rate of $|\frac{dT}{dt}| = 10.0\,°\text{C min}^{-1}$ (a) or $|\frac{dT}{dt}| = 0.1\,°\text{C min}^{-1}$ (b) plotted as in Fig. 6.11.

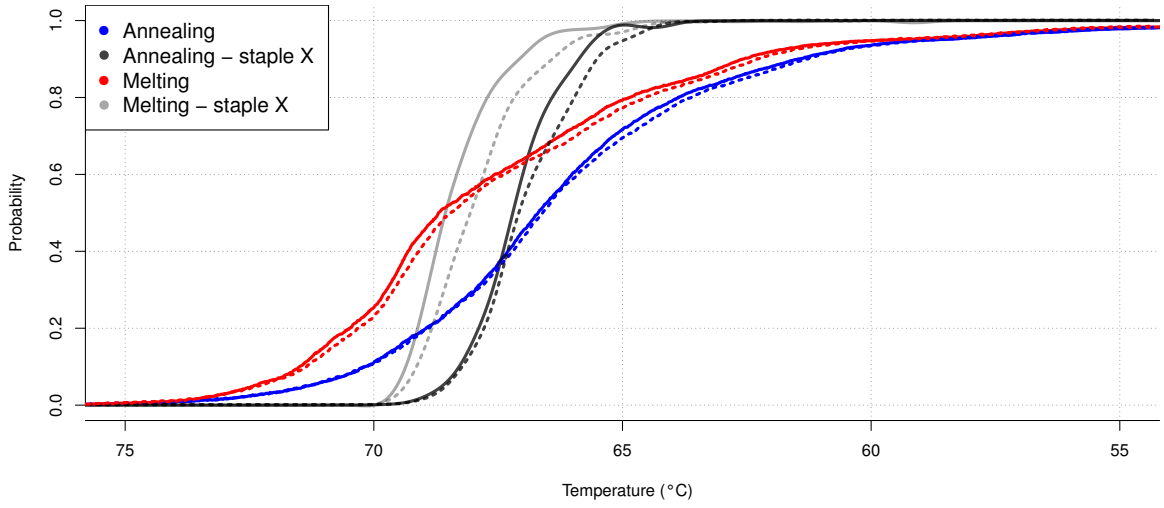


Figure 6.14: Comparison between the local and global model for $n = 2$, $\gamma = 2.5$. Solid lines correspond to predictions from the global model, while dashed lines are predicted by the local model. In blue and red: respectively annealing and melting predicted under the global model. In grey and black: smoothed annealing/melting curves for staple X.

6.8 Computing the required assembly time

Taking the model at face value, we compute the minimum required time to assemble a population of origami by considering the inter-arrival time for each staple. It is clear, intuitively, that a lower staple concentration leads to fewer encounters between staples and the scaffold. A well-formed origami requires exactly one of each staple to bind to the scaffold, and we want to compute the time required for 99% of the scaffold population to reach completion. It suffices to estimate the required time until the scaffold has encountered each staple, if we assume that the encounter between a staple and scaffold always leads to a successful binding and staples cannot unbind. We assume staples immediately bind with both domains, so internal bind rates are not considered here. This approach also discounts the possibility of two half-bound staples to occupy one domain each on the scaffold.

Staple binding is a bimolecular reaction for which we assume a well-mixed reaction vessel. The interarrival time $A(p)$ of a staple p at the scaffold is exponentially distributed as

$$A(p) \sim \text{Exp}(k_+[p]) \tag{6.64}$$

where $[p]$ is the concentration of the staple and $k_+ = 10^6\text{M}^{-1}\text{s}^{-1}$ is the bimolecular reaction rate constant. So, on average, the delay between the start of the folding and a staple of type p binding to the scaffold, is given as

$$E[A(p)] = \frac{1}{k_+[p]}. \tag{6.65}$$

Given a single-domain staple p and its associated concentration $[p] = 20n\text{M}$, we find $A(p) \sim \text{Exp}(0.020)$ and the expected inter-arrival time is 50 s. For an origami design with 76 two-domain staples and 14 single-domain staples, the time-to-completion T_c is a random variable given by

$$T_c = \max\left(\max_{i\in 1:76} B(p_i), \max_{i\in 1:14} A(p_i)\right) \tag{6.66}$$

where the distribution of $B(p_i)$ compensates for the double occurrence of binding sites on the scaffold so that $B(p_i) \sim \text{Exp}(0.040)$. The $90\%, 95\%$ and 99% percentile of the distribution of T_c is ≈ 4 min, $\approx 4\frac{1}{2}$ min and ≈ 6 min respectively. We conclude that a homogeneous and complete assembly of DNA origami requires in the order of 6 minutes, assuming a constant staple concentration of 20nM. Reducing the staple concentrations increases the required time.

6.9 Conclusion

We developed a model of origami assembly that enumerates internal loops in partially folded origami and estimates the cost of loop formation through a freely-jointed chain approximation. Two constants in the model are left as tuneable parameters, one being the scaling of the loop cost as a function of loop size and the other being the stabilizing contribution due to helical stacking. We identified a reasonable range of values of each parameter, and most combinations result in plausible model predictions. A variant of the model that is not energetically consistent was developed and found to closely approximate the behaviour of the original model.

Although the model is basic and void of steric constraints, it provides qualitative predictions of hysteresis and annealing temperature, indicating the time-scales in the model are reasonably well calibrated. The model provides support for reasoning about folding pathways; specifically, it provides a handle on the relation between staple incorporation and loop- and domain-stability, as well as the stochastic nature of DNA origami self-assembly. In the next chapter we utilize our model to control the pathway of a polymorphic origami tile.

Chapter 7

Folding pathways for a polymorphic tile

In this chapter we report on a novel type of DNA origami which, by design, does not fold into a single eventual shape. This polymorphic tile permits 74 separate configurations that each are fully-formed, and where each configuration corresponds to one of seven distinguishable shapes. Imaging the assembled tiles using AFM reveals that some well-formed tile shapes occur frequently, separate from a larger variety of 'broken' or 'incomplete' tiles. At this point we form two alternative hypotheses regarding the assembly process. The first hypothesis states that the distribution of shapes results from an assembly process that reaches an equilibrium, so that the distribution of shapes is a function of the thermodynamic stability of each fully assembled shape. In the second hypothesis, we assume each of the final folds are approximately equally stable, but the deciding factor in the final shape is the folding pathway. That is, the early interaction between staples and scaffold restrict the subsequently available shapes, and recovery from previous behaviour is hard.

In this chapter we apply the self-assembly model from Chapter 6 to the polymorphic tile, and find qualitative agreement between model prediction and experiment. The model suggests folding behaviour according to the second hypothesis, and to test this we modify the tile design to drastically alter the predicted folding pathway. The modifications to the original tile are small enough to avoid disturbing the relative differences in thermodynamic stability between folds. Imaging of the modified tile reveals a similar strong response in the experimentally observed shapes, which confirms the second hypothesis. This demonstrates control over the folding pathway of DNA origami in a way that was not demonstrated before, and validates the principles of origami self-assembly as given by the model of Chapter 6. It also provides intuitive understanding of self-assembly and complements existing studies into the nature of DNA self-assembly [22, 155, 154, 157]. More specifically, we characterise the folding pathway as

a cooperative assembly.

This chapter is organized as follows. First, we discuss the design of the polymorphic tile and provide a description of the state space of the model in Section 7.1. In Section 7.1.2 we define what constitutes a well-formed tile and classify the well-formed tiles according to shape. The simulation method is based on the self-assembly model of the previous chapter, which we discuss in Section 7.2. In Section 7.3 we describe the folding pathway for the polymorphic tiles and discuss modifications to the design that lead to a different distribution of shapes. The simulation code used to simulate the regular and the polymorphic tile is available at [2]. This chapter contains results from [10].

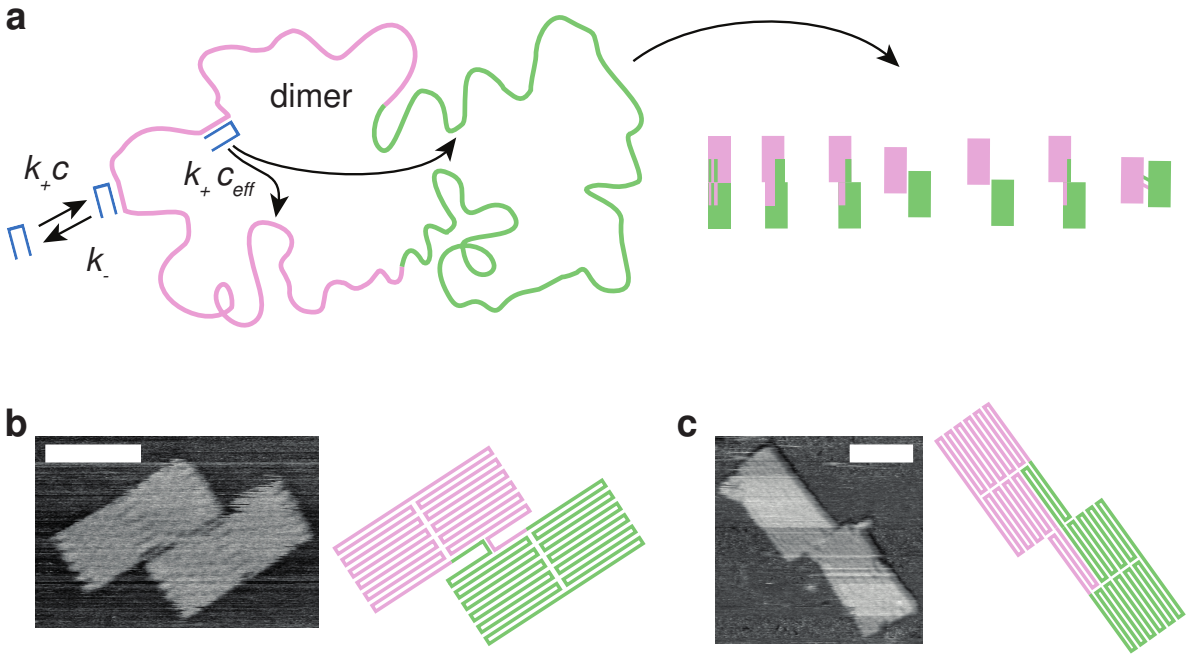


Figure 7.1: Adapted from [10]. The polymorphic tile. a) Joining two monomer scaffolds head-to-tail results in a tile that can adopt one of seven shapes. b) The polymorphic tile consists of two rectangular bodies joined along the side. Left: AFM imaging. Right: Scaffold routing. c) Same as (b) but the join occurs on the short side of the tiles. Scalebar: 50 nm.

7.1 Description of the polymorphic tile

We present a polymorphic version of the tile used in the previous chapter (Fig. 6.1), and in the remainder we call the tile from the previous chapter the 'monomer' tile. To create the polymorphic version, two copies of the monomer scaffold are merged head-to-tail into a single dimer scaffold. This design allows the scaffold and staples to self-assemble into several shapes, as in Fig. 7.1bc, and for this reason it is called a polymorphic tile.

Both the monomer and the polymorphic tile are designed and synthesized by Dr. Kather-

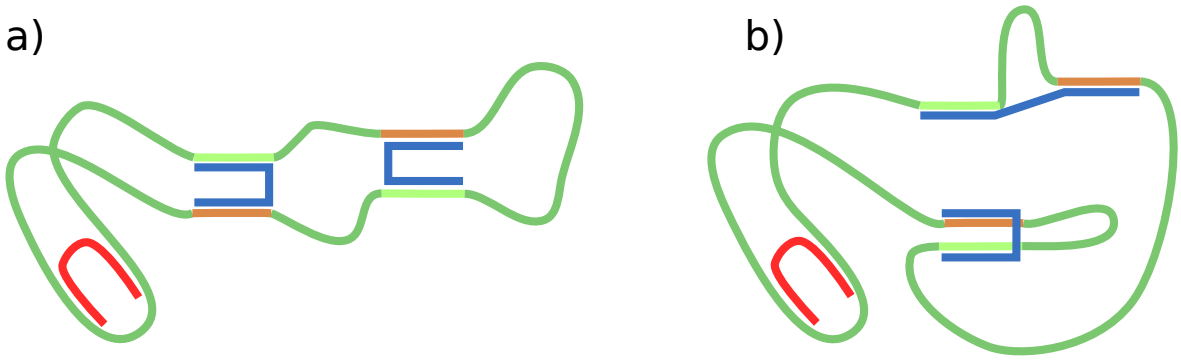


Figure 7.2: The dimer scaffold pictured with one red staple and two identical blue staples. Two identical and fully bound staples can be incorporated in exactly two ways: compare a and b. The scaffold domains complementary to the blue staples are highlighted in light green and orange.

ine E. Dunn from pUC19 plasmid, who also performed AFM imaging of the tiles. The thesis by Dunn is an excellent reference on the polymorphic tile [168], and we highlight relevant details of the assembly protocol. Prepared dimer scaffold, at $4 - 10$ nM concentration, was assembled in the presence of staples at 10x excess concentration by cooling from 95 °C to room temperature at a rate of 1 °C min^{-1} in a buffer containing 12.5mM magnesium acetate and 40mM Tris-acetate (pH 8.3). By comparison, all our simulations assume a constant staple concentration of 20 nM and equal buffer conditions, and are run at a temperature gradient of 0.4 °C min^{-1}. The scaffold concentration is assumed to be negligible.

A fully-assembled polymorphic tile consists of one dimer scaffold and two copies of each staple: there are 76 types of two-domain staples and 14 types of single-domain staples. A dimer scaffold is a single circular scaffold that repeats the monomer sequence twice, and as a result has repeated domains that are exactly the length of a monomer scaffold apart. Examples of the dimer scaffold routing are given in Fig. 7.3 and Fig. 7.4. Two identical two-domain staples, each hybridized to the dimer scaffold with both domains, can occur in one of two configurations (states), where we discount the possibility of scaffold entanglement. In Fig. 7.2 two dimer scaffolds are depicted, each incorporating a single red staple and two identical blue staples. The two configurations are not isomorphic, because the scaffold nucleotide sequences (if not length) of the formed loops are different in each configuration. As a result, there are 2^{76} combinations in which two of each of the 76 staples could be hybridized to the scaffold. Many of these configurations are inaccessible due to steric constraints, and we discuss an approximation to the steric constraints for use with the self-assembly model in Section 7.2.1.

Some folds are flexible, so that a range of closely related shapes would be observed by AFM imaging, rather than a single shape, and this must be considered when comparing the model predictions with AFM imaging. Specifically, the scaffold routing of Fig. 7.3c allows the two

halves to rearrange somewhat freely: this is less true for the scaffold routings in Fig. 7.3abd.

7.1.1 State space

For each type of two-domain staple there are 34 distinct patterns of domain binding (states), with between zero and four copies of the staple bound to the dimer scaffold. We enumerate the possible states by conditioning on the number of staples attached to the dimer scaffold as follows:

- The first possibility is an empty scaffold without any attached staples.
- Given one attached staple, there are four states in which the staple is half-bound and four states in which the staple is fully bound.
- Given two attached staples, there are six states in which both staples are half-bound, eight states with one half-bound and one fully bound staple, and two states with two fully bound staples.
- Given three attached staples, there are four states with three half-bound staples and another four states with one fully bound and two half-bound staples.
- Finally, there is the possibility that four half-bound staples are attached to the scaffold.

For a single-domain staple and the associated pair of scaffold domains there are four states, as each domain is hybridized independently of the other.

The above enumeration ignores rotational symmetry, which reflects the approach used in the simulation code, but this bears no consequence on the correctness of the simulation. We therefore distinguish between $34^{76} \times 4^{14}$ states during the assembly of the polymorphic tile. There exist 2^{76} states for which each two-domain staple is fully bound. Formally, the state space is given as $S = \mathsf{p}_1 \times \mathsf{p}_2 \times \ldots \times \mathsf{p}_{90}$, where p_i denotes the set of possible states for the i-th staple.

7.1.2 Definition of well-formed tiles

The eventual fold of the tile is classified in the model as either well-formed or misfolded. A well-folded tile consists of two monomer tiles joined via the scaffold, for example see Fig. 7.1bc, and we define:

Definition 7.1.1. *An dimer tile is well-formed if*

1 No part of the scaffold is left unhybridized, and

2 Each staple occurs exactly twice in the structure.

Note that each requirement can be satisfied independently of the other. Because well-formed tiles have the maximum number of possible base-pairs, and contain no unhybridized sections, we hypothesize that well-formed tiles are very close to the thermodynamic optimum, and assume that the thermodynamic stability of each well-formed dimer tile is approximately equal, despite variations in shape.

Various events may prevent the scaffold from reaching a well-formed fold. Importantly, the polymorphic tile has modes of failure that are not found in regular origami. For example, certain combinations of staple joins can entangle the scaffold. Another mode of failure are incompatible partially folded segments of the tile (see Fig. 7.4e). In this light the observation of seemingly well-formed tiles is remarkable in itself (see Fig. 7.1bc).

An interesting question now is: given an arbitrary dimer scaffold, how many well-formed structures exists, given the steric constrains of the molecular geometry? What if we allow more general scaffolds? In this thesis, no satisfying answer to this question is found. Instead, we merely demonstrate that 74 different scaffold routings of the dimer tile exists that result in well-formed tiles, while respecting the natural constraints arising from the molecular geometry.

We now classify the well-formed tiles according to shape. To start, we assign each staple a type according to their position in the tile. The first group are the single-domain staples, grey in Fig. 7.3, which occupy the sides of the tile. The other staples exclusively consist of two domains, and are classified as either body staples or seam staples. The seam staples are centrally located in the design, shown as dark orange, and come in five pairs. The other two-domain staples are not centrally located and are called body staples, shown in blue.

To classify each shape, we describe the state of seam staples. The fold m:n has m pairs of seam staples that connect domains within the same half of the scaffold, and n pairs of seam staples that form connections between domains on opposite halves. The folds m:n and n:m are of equal shape and cannot be distinguished in the experiment, so we end up with three types: 3:2, 4:1 and 5:0, respectively a, b and c in Fig. 7.3. The type 5:0 comes in four variants; i, ii, iii, iv, that indicate varying levels of offset between the two halves (respectively ci, cii, ciii, civ in Fig. 7.4). In addition, there is the possibility that the scaffold routing is not planar, but has crossovers, and such folds are called N.P. for 'not planar' (Fig. 7.3d). Folds that are not well-formed, as in Fig. 7.4e, are referred to as 'N.T.' for 'no type'. At the time of writing there is no construction available to demonstrate that only the just mentioned shapes satisfy both requirements to be 'well-formed', but based on experimental evidence we believe the list is exhaustive.

We now explain how many states correspond to each of the well-formed shapes, which we call the multiplicity of the shape. Considering the 5:0i shape, each long (32-nt) single domain staple

can occur either on the central join between the halves or on the outer edge (see Fig. 7.3c), and the multiplicity of the shape 5:0*i* is two. For the shapes 5:0*ii*, 5:0*iii*, 5:0*iv* (Fig. 7.4c) the join of the two tiles occurs near one of the corners of each half of the structure. Since the corners are distinguishable from one another by scaffold sequence, the multiplicity of these shapes is four. The non-planar folds are unique in having multiple crossings of the scaffold, and for this reason no planar embedding of the associated graph is possible. These folds have either three (Fig. 7.4di) or five (Fig. 7.4dii) crossovers of the scaffold, where a similar corner-specific symmetry applies. We list the multiplicity of each shape in Table 7.1.

Shape	Multiplicity	Details	$\mathsf{H}(s)$ planar?
5:0*i*	2	top/bottom variants	Yes
3:2, 4:1, 5:0*ii*, 5:0*iii*, 5:0*iv*	4	corner variants	Yes
N.P.	52	$2 \times \left(\binom{6}{3} + \binom{6}{5}\right)$, left/right $\times$ position of crossover	No
Sum	74		

Table 7.1: Each well-formed shape corresponds to not just one, but several different scaffold routings (states), which we call the multiplicity of the shape.

7.2 Model

To simulate the self-assembly of the polymorphic tile we employ the self-assembly model of Chapter 6, which we augment with the exclusion algorithm of Section 7.2.1. We first discuss the model parameterisation.

We employ the 'local' self-assembly model with basic parameterisation, this being the volume-exponent $\gamma = 1.5$ and no energetic contribution from end-to-end stacking of helices ($\Delta G^{\text{stack}} = 0$). Distinct from the model description in Chapter 6, we set the domain stability to be independent from the domain nucleotide sequence. For regular domains ΔG^{duplex} is equal to the predicted stability averaged over all possible sequences of 16 base-pairs, and for double-length domains it is averaged over all possible sequences of 32 base-pairs.

A large number of intermediate states admit no planar embedding of the associated graph, and these states cannot be ruled out based on steric requirements. In addition the well-formed shapes of type N.P. also do not permit planar graph embedding. Therefore, we use the local variant of the self-assembly model, rather than the global variant which currently does not allow states with non-planar embedding.

Parameterisation

To obtain a simulated distribution of shapes, we generate 1600 trajectories in the model, using an annealing rate of $0.4\,^{\circ}\text{C min}^{-1}$. For each trajectory, we check if the final state is equal to one

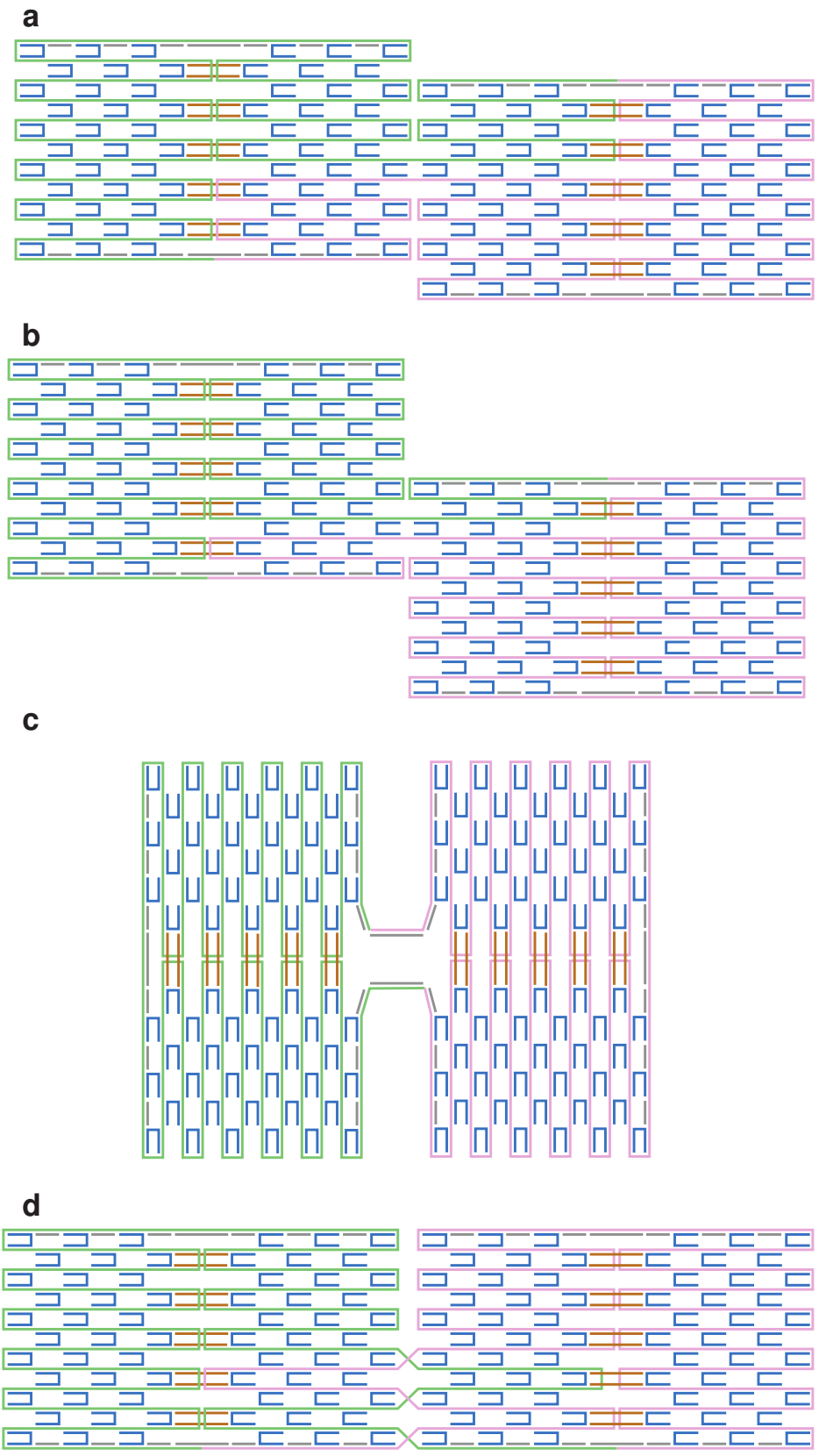


Figure 7.3: Four possible folds of the polymorphic tile, each having an unique scaffold routing and staple orientations. In blue: body staples. In orange: seam staples. In grey: single-domain staples. The scaffold is in green/pink. These shapes are classified as 3:2 (a), 4:1 (b), 5:0i (c) and non-planar (N.P.) (d).

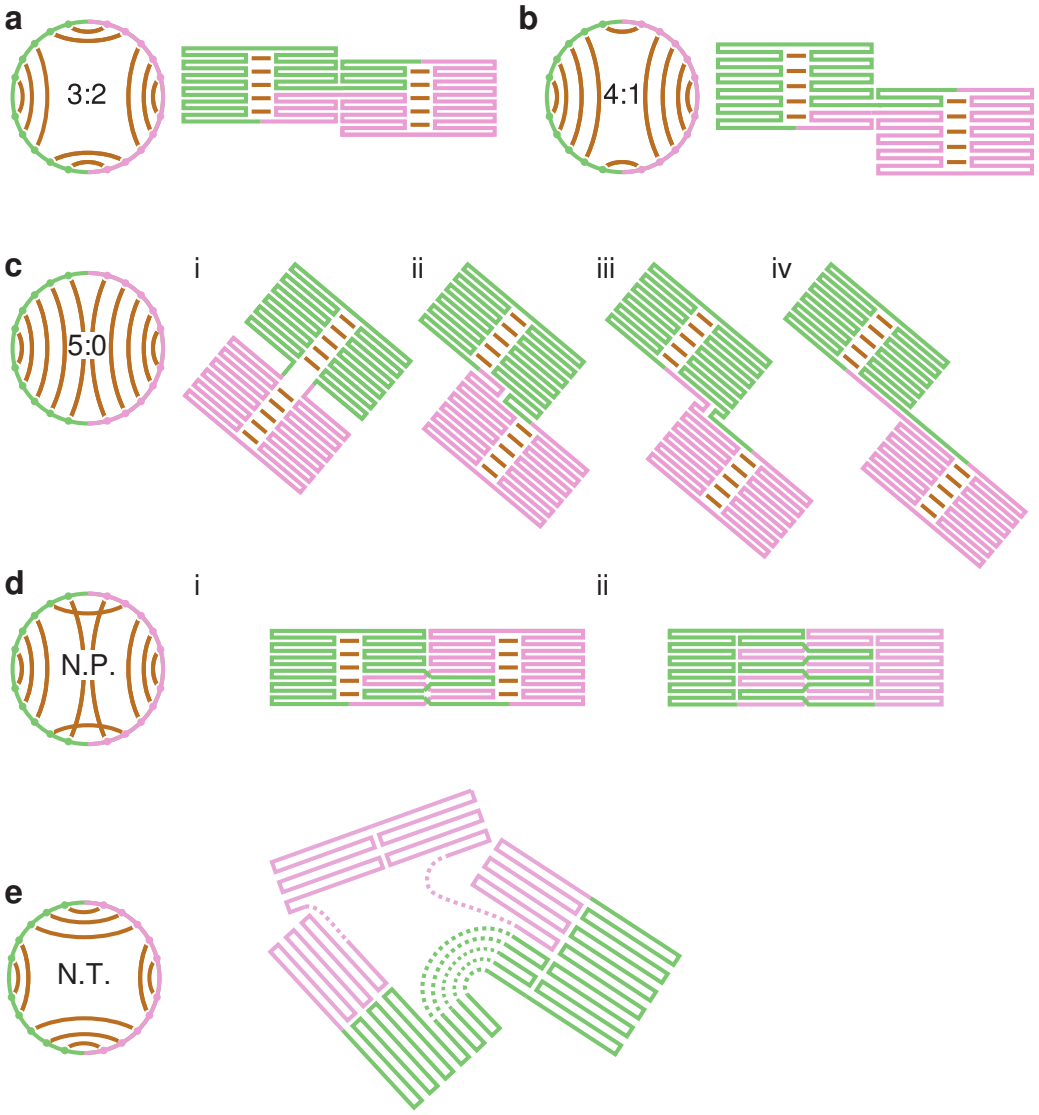


Figure 7.4: Adapted from [10]. Different folds for the polymorphic origami tile and the corresponding seam schematic. a-c) These folds are distinguished by considering the folding pattern of the seam staples, leading to classifications 5:0, 4:1 and 3:2 respectively. Shape 5:0 is subdivided into four shapes depending on offset, 5:0*i*, 5:0*ii*, 5:0*iii*, 5:0*iv*, depicted respectively as ci, cii, ciii and civ. d) A fold where the two halves are joined with zero offset is possible when multiple crossovers occur in the scaffold routing. Folds with three (di) or five (dii) crossings are possible. e) This polymorphic tile consists of three well-formed components, but as a whole, the tile is not well-formed because single stranded sections between the components remain.

of the 74 well-formed states (cf. Table 7.1), and if so, label it with the appropriate shape. If it is not a well-formed tile we report it as N.T. (no type). The resulting distribution of shapes is sensitive in the temperature ramp dT/dt, but uncertainty in the experimentally observed distribution prevents meaningful fitting to data. The cooling rate of $0.4\,^{\circ}\mathrm{C\ min}^{-1}$ was chosen from one of three rates, $0.1\,^{\circ}\mathrm{C\ min}^{-1}, 0.4\,^{\circ}\mathrm{C\ min}^{-1}$ and $1.0\,^{\circ}\mathrm{C\ min}^{-1}$, by way of comparison to the empirical distribution found in the unmodified experiment. This procedure puts the model at risk of overfitting dT/dt, which we justify as follows. Firstly, each rate in the model scales in the bimolecular rate constant k_+, so that k_+ is an effective time-scale in the model, alongside dT/dt. As a result the sensitivity in dT/dt is equivalent to the sensitivity in k_+. Because the value of $k_+ = 10^6\mathrm{M}^{-1}s^{-1}$ is applicable within an order of magnitude, a conservative fitting procedure of dT/dt is in order. Secondly, further uncertainty in the rates is due to staple concentration: the setting for the simulation (all staples at 20 nM) is lower than that in experiment (4-10 nM scaffold in presence of 10x staple excess). Thirdly, we have chosen a value from a highly restricted set of options. No further fitting was performed.

7.2.1 Exclusion algorithm

In the model, each state describes a particular set of staple binding behaviour. A specific type of staple is, for example, present twice on the scaffold in a given orientation, or perhaps not at all. Given a fully-bound two domain staple, the two hybridized scaffold domains are held together within a few tenths of a nanometre of each other. The steric requirements imposed by each staple, however, cannot be satisfied in many states. This problem does not occur for the monomer tile. We develop an algorithm to prevent the model from accessing these infeasible states. This method yields an approximation to the real steric constraints: it does not guarantee that each legal state satisfies the constraints or that all states that satisfy the steric constraints are legal. The simulation method from Section 6.5 still applies, but a non-zero transition $\mathbf{R}(s, s') > 0$ is removed from the model if the target state s' is illegal. We now describe the set of legal states.

Define a connected component of an origami as a set of hybridized scaffold domains such that each domain can be reached from each other domain without leaving the set. Two scaffold domains hybridized to the same staple are defined to be connected, as are two adjacent scaffold domains hybridized to different staples. We now define the set of connected components in an origami in terms of the graph embedding. As in Section 6.5, each state has a graph-representation

$$\mathsf{H}(s) = (\mathsf{V}, \mathsf{E}(s)) \tag{7.1}$$

so that the vertices are the joins between domains on the scaffold and edges represent domains between the appropriate vertices and staple-induced links between non-adjacent vertices. A

labelling function

$$\mathsf{L} : \mathsf{E} \to \{\text{single-stranded}, \text{double-stranded}, \text{crossover}\} \tag{7.2}$$

assigns the status of each edge, and we define a derived graph $\mathsf{H}'(s) = (\mathsf{V}, \mathsf{E}'(s))$ by dropping the edges that represent unhybridized scaffold domains

$$\mathsf{E}'(s) = \Big\{ e \in \mathsf{E}(s) \Big| \mathsf{L}(e) \in \{\text{double-stranded}, \text{crossover}\} \Big\}. \tag{7.3}$$

$\mathsf{H}'(s)$ is a non-directed graph where edges induce an equivalence relation $\sim_R$ on the vertices that indicates if a path exists between any two vertices. Then the set of connected components $\mathsf{C}(s) = \{(\mathsf{V}_i, \mathsf{E}_i)\}_i$ is equal to the set of equivalence classes of vertices (and their associated edges) under this relationship,

$$\{\mathsf{V}_i\}_i = \mathsf{V}/\!\sim_R \tag{7.4}$$

$$\mathsf{E}_i = \Big\{ (v_1, v_2) \in \mathsf{E}'(s) \Big| v_1, v_2 \in \mathsf{V}_i, \Big\}. \tag{7.5}$$

A segment of origami is considered stress-free if it occurs as subset of a well-formed two-dimensional shape (see Fig. 7.4). Let S_w be the set of 74 states that correspond to a well-formed shape (see Table 7.1). These pre-defined folds satisfy the constraints imposed by finite staple length and steric exclusion. A partially folded origami s is legal if and only if each of the components are stress-free, that is

$$\forall\, (\mathsf{V}, \mathsf{E}) \in \mathsf{C}(s) \quad \exists s_w \in S_w : \mathsf{E} \subseteq \mathsf{E}(s_w). \tag{7.6}$$

As the simulation moves from state to state, a representation of the connected components is updated with each transition.

Misfolds are said to occur in the model when at least two connected components would be incapable of satisfying the constraints if they were to become connected to one another. At that point, folding cannot advance unless one of the components unfolds, allowing another to expand. Figure 7.4e shows a misfolded dimer that has three connected components that can join to form a stress-free state. When simulating the unmodified polymorphic tile, about half of the simulations end in a misfolded (N.T.) state.

7.2.2 Example rate calculations

Following the model, the rate of a half-bound staple binding to the scaffold via its unbound domain is given by $k_+ c_{\text{eff}}$ where c_{eff} is the effective concentration of the opposing domain and

given as (Eq. 6.38):

$$c_{\text{eff}} = \frac{1}{N_A}\left(\frac{3}{2\pi E[\mathsf{R}^2]}\right)^{3/2} \tag{7.7}$$

where $E[\mathsf{R}^2]$ is the weight of the newly formed loop between appropriate vertices, found by Dijkstra's shortest path algorithm, plus a segment of length λ_{ss} representing the newly formed link that closes the loop (Eq. 6.47). We now give rate computation examples for the dimer scaffold, where a half-bound staple can bind to a nearer and further available complementary scaffold domain.

Consider the half-bound staple shown in Fig. 7.5a that is hybridized to an otherwise empty scaffold. A seam staple, labelled A, is used as an example here. Its second domain hybridizes to either of two sites: a closer site is connected by a 448-nt single-stranded DNA chain ($E[\mathsf{R}^2] = 480$ nm^2) and the more distant site is connected by a composite chain comprising a 2208-nt single-stranded chain and one rigid 16-bp double-stranded segment ($E[\mathsf{R}^2] = 2400$ nm^2). Following the computation we find the effective transition rate between s_0 and s_1 to be $\mathbf{R}(s_0, s_1) = 51$ s^{-1}, and for the alternative the option of forming the longer loop the rate is 4.5 s^{-1}. In this case the staple is 11 times more likely to bind to the closer domain.

Binding of one staple affects the binding of others by changing the characteristics of the sections of partly-formed origami that link their two binding domains. We now compute the loop cost and hybridization rate for a second seam staple, staple B, in the presence or absence of staple A. In the absence of staple A, the shorter of the two loops that connect two binding domains of the second staple consists of a 864-nt single-stranded DNA chain: $E[\mathsf{R}^2] = 990$ nm^2, and the effective transition rate is 18 s^{-1}. In the presence of staple A, the loop passes through the link formed by staple A and comprises 384 nt single-stranded DNA, 3 rigid 16-bp double-stranded DNA segments and a staple crossover modelled as a single segment of length λ_{ss} (Fig. 7.5b): for this shortened loop, $E[\mathsf{R}^2] = 530$ nm^2 and effective rate $\mathbf{R}(s_2, s_3) = 44$ s^{-1}. Inserting staple A increases the rate of hybridization of the second domain of staple B by a factor of 2.5 by shortening the distance between its binding sites. Further details on the computation are found in Table 7.2.

7.2.3 Treatment of paired seam staples

The paired seam staples (orange, Fig. 7.3) represent a somewhat special case in the model. When one of the pair of staples is fully bound to the scaffold, it brings the hybridization sites for the other staple in very close proximity. The situation where both seam staples are fully bound results in a particularly small loop consisting of exactly two segments of length λ_{ss}, so

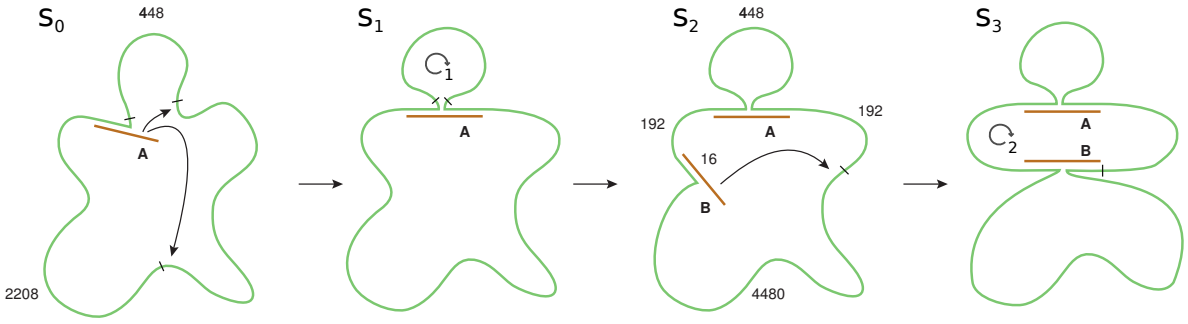


Figure 7.5: Adapted from [10]. Example rate calculations for the polymorphic tile. The presence of staple A increases the stability of staple B.

Loop	Weight	$E[\mathsf{R}^2]$ /nm^2	c_{eff} /μM	ΔG /kcal mol^{-1}
loop-1	$0 \times \lambda^2_{16,ds} + (28 \times 16/3 + 1) \times \lambda^2_{ss}$	487.1	50.9	6.54
alternative to loop-1	$2 \times \lambda^2_{16,ds} + (138 \times 16/3 + 1) \times \lambda^2_{ss}$	2447	4.52	8.14
loop-2	$4 \times \lambda^2_{16,ds} + (24 \times 16/3 + 2) \times \lambda^2_{ss}$	539.6	43.7	6.64
loop-2, without A	$2 \times \lambda^2_{16,ds} + (54 \times 16/3 + 1) \times \lambda^2_{ss}$	995.5	17.7	7.25

Table 7.2: Adapted from [10]. Composition, weights, effective concentration and energetic cost for the loops of Fig. 7.5. ΔG is given for $T = 60\,^\circ$C.

that $E[\mathsf{R}^2] = 2\lambda^2_{ss}$ for this loop. The predictions of the model remain physically sensible: a second staple binding to a seam has an overall ΔG which is ≈ 4.4 kcal mol^{-1} less favourable (at $T = 60\,^\circ$C) than a continuous duplex. This destabilization is equal to that expected from a 5-nt bulge within a duplex.

This cooperative effect is destroyed when one of the seam staples is removed from the input set of staples. In this case the model predicts incorporation temperatures for the unbroken staple that are lower than the regular case by 2.0 °C, compared to 2.2 °C observed in experiment through application of FRET measurements [10]. It is therefore clear that we do not overestimate the cooperative stabilisation of seam staples.

7.3 Folding pathway

The predictions of the model are compared to experimental observation of the tiles and we validate our understanding of the pathway through modifications to the experiment. Initially, we discuss the folding pathway for the unmodified polymorphic tile.

Given detailed AFM imaging, such as in Fig. 7.1bc, we are able to unambiguously classify the assembled tile as one of the pre-determined shapes. To estimate the distribution of shapes, however, many observations are required. For this reason the field of view is adjusted to contain not just one tile, but many of them, and subsequently the resolution per tile diminishes, making

Design	N.P.	3:2	4:1	5:0i	5:0ii	5:0iii	5:0iv	N.T.
Original	65	268	207	162	32	33	4	829
Broken seam	2	70	60	955	238	222	38	15
Elongated staples	8	602	106	216	89	16	4	559
Alternative seam	1	1517	2	10	1	3	0	66
Lower-right	113	187	190	119	30	23	182	756

Table 7.3: Predicted shape distribution for the polymorphic tile. Each design is simulated at $|dT/dt = 0.4|\,^{\circ}\mathrm{C\ min}^{-1}$ for a total of 1600 paths.

an unambiguous classification more difficult. To resolve this, a fitted value for the relative offset of the two halves of the tile is reported [168]. The shapes predicted by the model are equally classified by relative offset to allow a direct comparison (Fig. 7.6). Only objects with a surface area close to what is expected for a well-formed tile are considered for offset fitting [10]. Of the considered objects, 44% obtained a successful fit. This is not to say 66% of the tiles were misfolded. Because the tile is flexible, well-formed tiles can twist prior to fixating to the mica surface. In the following we discuss the simulation results, experimental results and folding pathway of five variants of the polymorphic tile:

- The unmodified tile
- Broken seam modification
- Elongated staple modification
- Alternative seam modification
- Lower-right modification

The simulation output for all five variants of the experiment is found in Table. 7.3.

7.3.1 Seam-mediated assembly

The folding pathway is expected to be dominated by the behaviour of the body staples. In isolation, the loops closed by these staples are smaller than those closed by the seam staples, meaning body staples are generally more stable than seam staples early in the assembly. In addition, the body staples outnumber the seam staples roughly seven-to-one. If we assume each body staple binds to the nearest location along the scaffold (ignoring the presence of other staples), the fold necessarily ends up as 5:0i, as in Fig. 7.3c, because every other fold requires at least some body staples to bind to the further rather than the nearer location.

Contrary to that intuition, the model predicts, and the experiment confirms, that each of the 5:0, 4:1 and 3:2 shapes occur approximately with equal ratio (Fig. 7.6a). This is explained by the *pairing* of the seam staples. The cooperative effect between the pair of seam staples is significant, because the presence of one seam staple brings the binding sites of the paired staple in close proximity, which we also discuss in Section 7.2.3. The presence of one body staple, in

contrast, also promotes the binding of other body staples, though to a lesser extent. Because of the increase in stability once both seam staples are in place, the connections mediated by the paired seam are less likely to unbind, when compared to the scaffold joins mediated by body staples.

The stabilizing effect of the seam staple pairing is observed in the simulation by counting the number of re-orientation events for each staple type. In the polymorphic tile, each two-domain staple binds to the scaffold in one of two possible orientations, as depicted for the blue staples in Fig. 7.2. A reorientation event is counted each time the orientation changes, and is counted separately for each type of staple. In the example of Fig. 7.2, the simulation counts one re-orientation event if the state changes from a to b or vice versa. In Fig. 7.7a the average number of reorientation events during assembly is depicted for each staple type, and shows that seam staples are less likely to reorientate during the folding.

The early incorporation of seam staples, and their increased stability when compared to the body staples, disfavours the formation of the 5:0 shape. Staples are more likely to stay hybridized to nearer domains (Section 7.2.2), because binding to nearer domains results in shorter loops and hence in reduced thermodynamic cost. The 5:0 shape requires the seam staples to join between domains that are spaced far apart along the scaffold, and assuming an otherwise empty scaffold, these connections are less likely to form. The relatively low rate of reorientation by the seam staples further suggests that body staples, although vastly outnumbered by the seam staples, are effectively steering the folding away from reaching the 5:0 shape. We illustrate the folding pathway in Fig. 7.8.

To further demonstrate the importance of seam behaviour in the model, we show that the early seam staple behaviour is highly indicative of the eventual shape. On average, the seam-staples are incorporated slightly earlier than body staples. The annealing temperature for a two-domain staple is given by the first temperature at which, on average, at least one staple is bound to the scaffold with both domains (so that the average occupancy for that type of staple is $\geq 50\%$). For one set of 1600 paths, the following temperatures were found:

- The average for seam-staples was 64.21 °C
- The average for body-staples was 63.91 °C
- The average over all two-domain staples was 63.95 °C.

Given the temperature gradient of 0.4 °C min^{-1}, we find that seam staples attain 50% occupancy about 3/4-ths of a minute earlier than body staples. Some body staples are, however, incorporated earlier on average, and some later, as depicted in Fig. 7.9. The order depends on the position in the origami, because the loops closed by each staple are of different lengths, meaning some staples are destabilized more than others. To demonstrate the influence of staple

behaviour on the eventual fold, we test for three properties at $T = 64.21\,^{\circ}\mathrm{C}$, the average annealing temperature of seam staples. Each property is typical to one of the 5:0, 4:1, or 3:2 folds. The properties that we test for are:

1 A 140-domain seam link with no cross-link to the other half of the scaffold, characteristic of fold 5:0.
2 A 112-domain link with a shorter cross-link, characteristic of fold 4:1.
3 Two 56-domain links, including one internal link and one cross-link between halves of the scaffold, characteristic of fold 3:2.

The properties are mutually exclusive, so that any dimer tests positive for at most one property. Satisfying one of the properties is highly indicative of eventually reaching the associated fold, as we see in Fig. 7.10. Additionally, the correlation between seam pairs shows a distinct pattern at early stages of the folding, which is further evidence of the influence of seam staple behaviour on the final shape (Fig. 7.11).

We now describe two adaptations to the original design, dubbed ‘broken seam’ and ‘extended staples’. Both are designed to alter the folding pathway without changing the relative stability of the well-formed shapes.

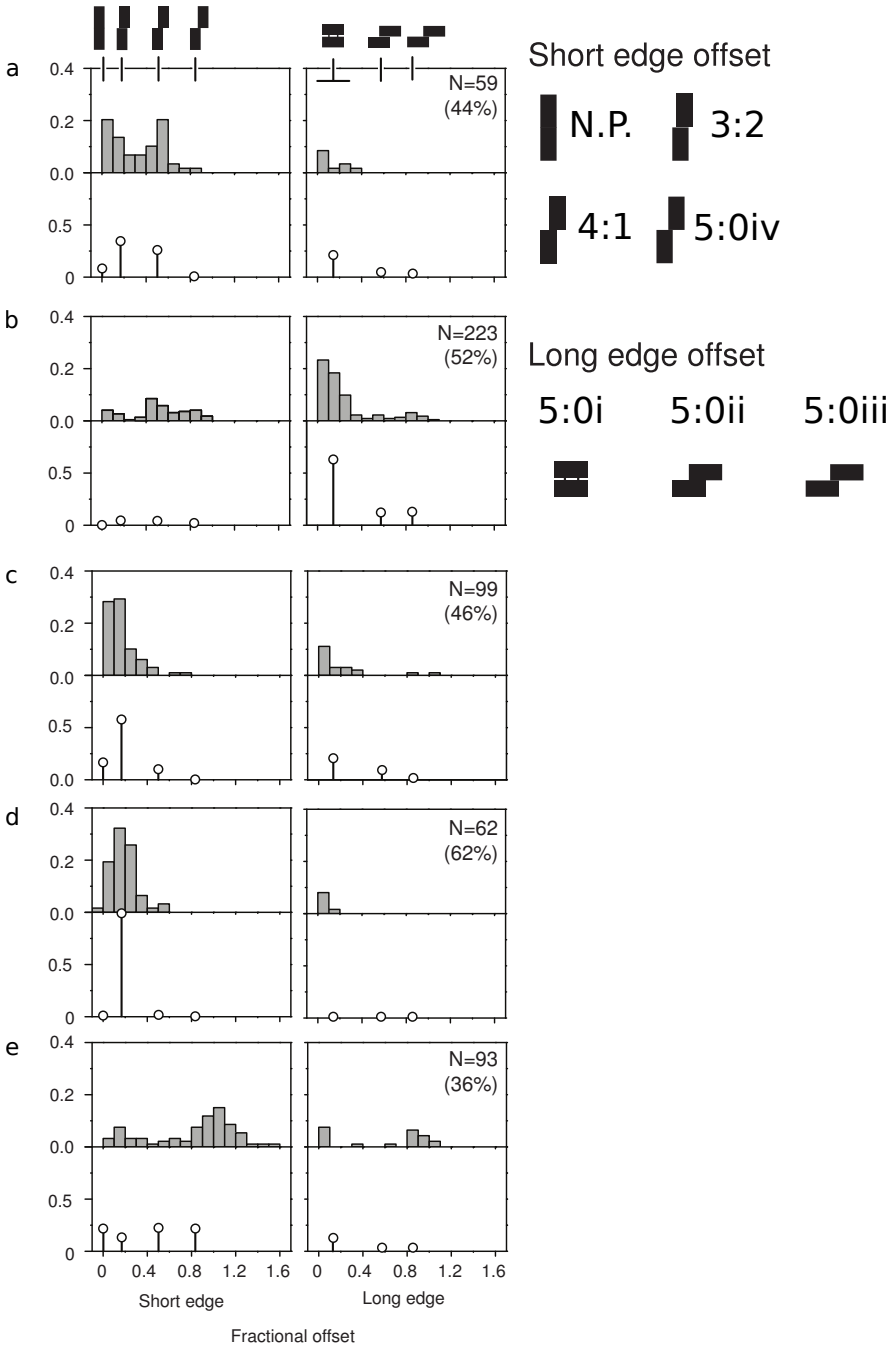


Figure 7.6: Adapted from [10]. Imaging obtained from AFM was processed to identify objects of approximately the predicted size of a well-formed tile. An automated fit determines the fractional offset that occurs either on the long edge or the short edge of the tile. The number of fitted tiles (N) and an estimate of the fraction of well-formed tiles in the experiment is displayed on the top right Lower half: matching predictions from the model by simulating 1600 paths at $0.4\,^{\circ}\mathrm{C\ min}^{-1}$. a) Original polymorphic tile design. b) Broken seam modification. c) Elongated staple modification. d) Alternative seam modification. e) Lower-right modification.

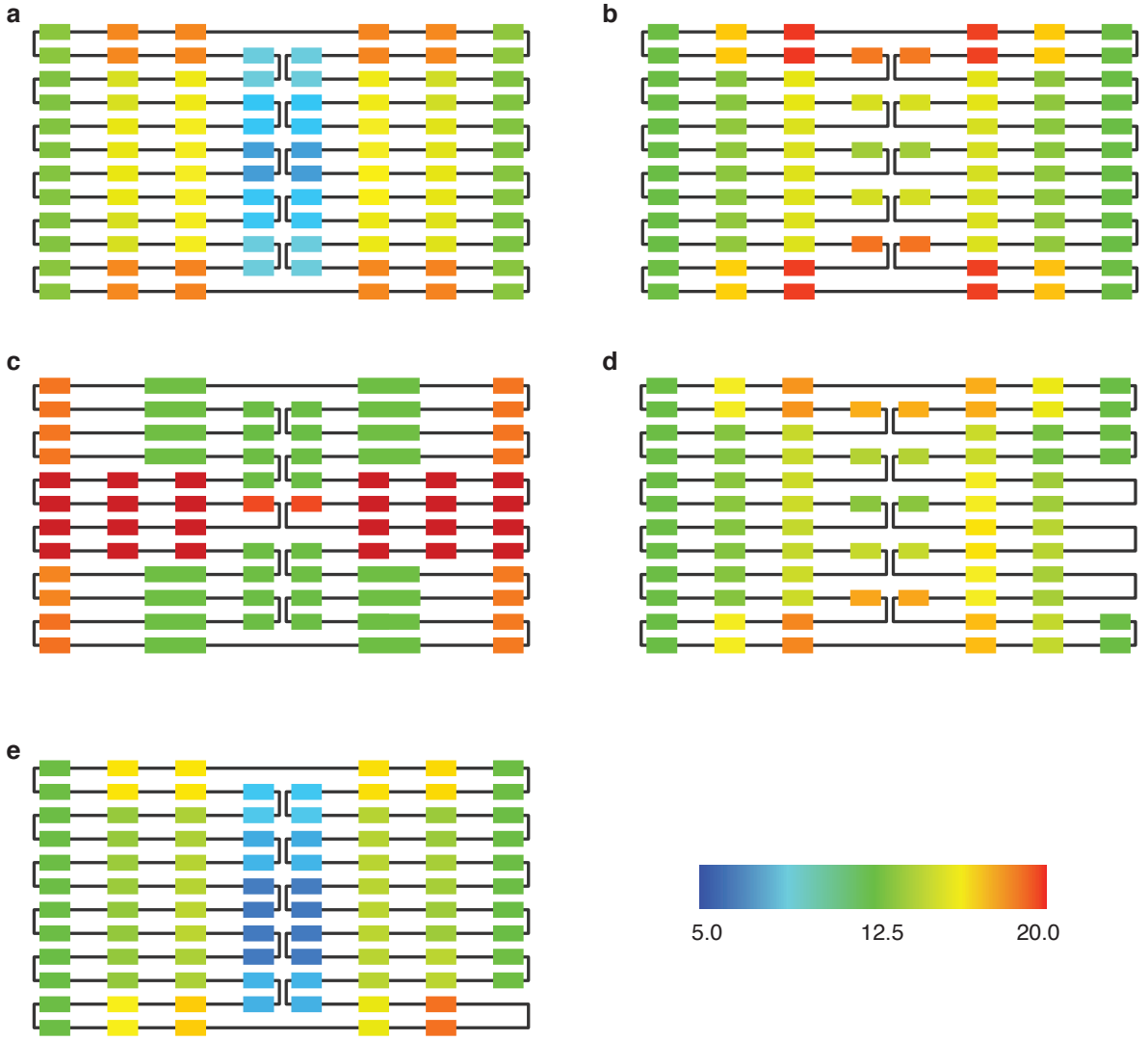


Figure 7.7: Adapted from [10]. The number of reorientations for each staple type during annealing, averaged over 1600 runs. A reorientation occurs when a fully-bound, two-domain staple unbinds from the scaffold at one domain, and then binds at the alternative binding site. Bar: Reorientation counts between 5.0 - 20.0 are coded by colour. Staples with less than 5.0 reorientation events on average plotted dark-blue and staples with more than 20.0 reorientation events on average are plotted dark-red. a) The original staple set. b) The broken seam modification. c) The elongated staple modification. d) The alternative seam modification. e) The lower-right modification.

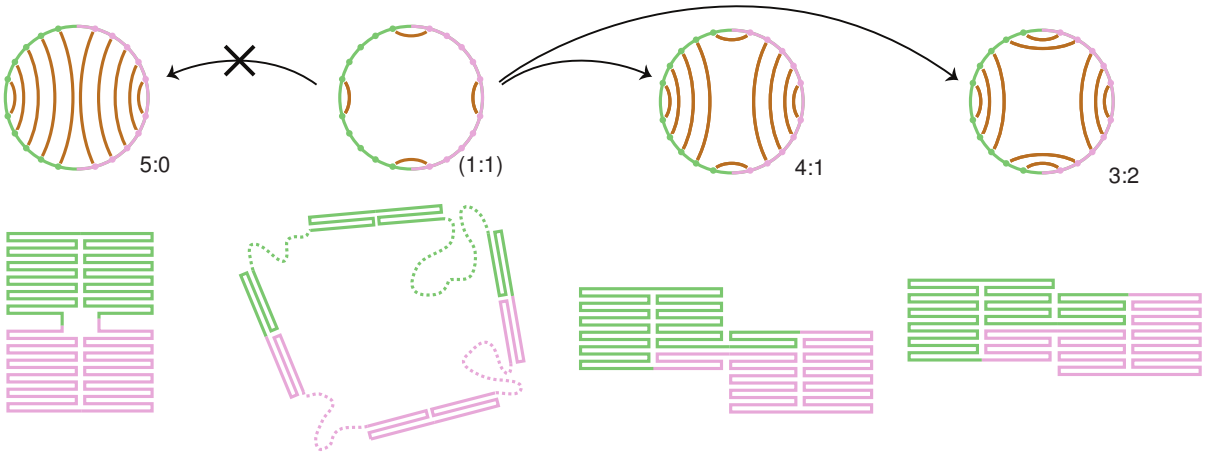


Figure 7.8: Adapted from [10]. Idealized folding pathway for the polymorphic tile. Seam staples mediate relatively stable long distance scaffold connections. Seam connections are more likely to form between nearer domains, ultimately suppressing the formation of 5:0. In the figure, the intermediate fold 1:1 is formed by inserting four pairs of seam staples in such a way that minimizes the length of the newly formed loops. This intermediate state can still fold into shapes 3:2 and 4:1, but cannot form 5:0 without breaking previously formed seam connections.

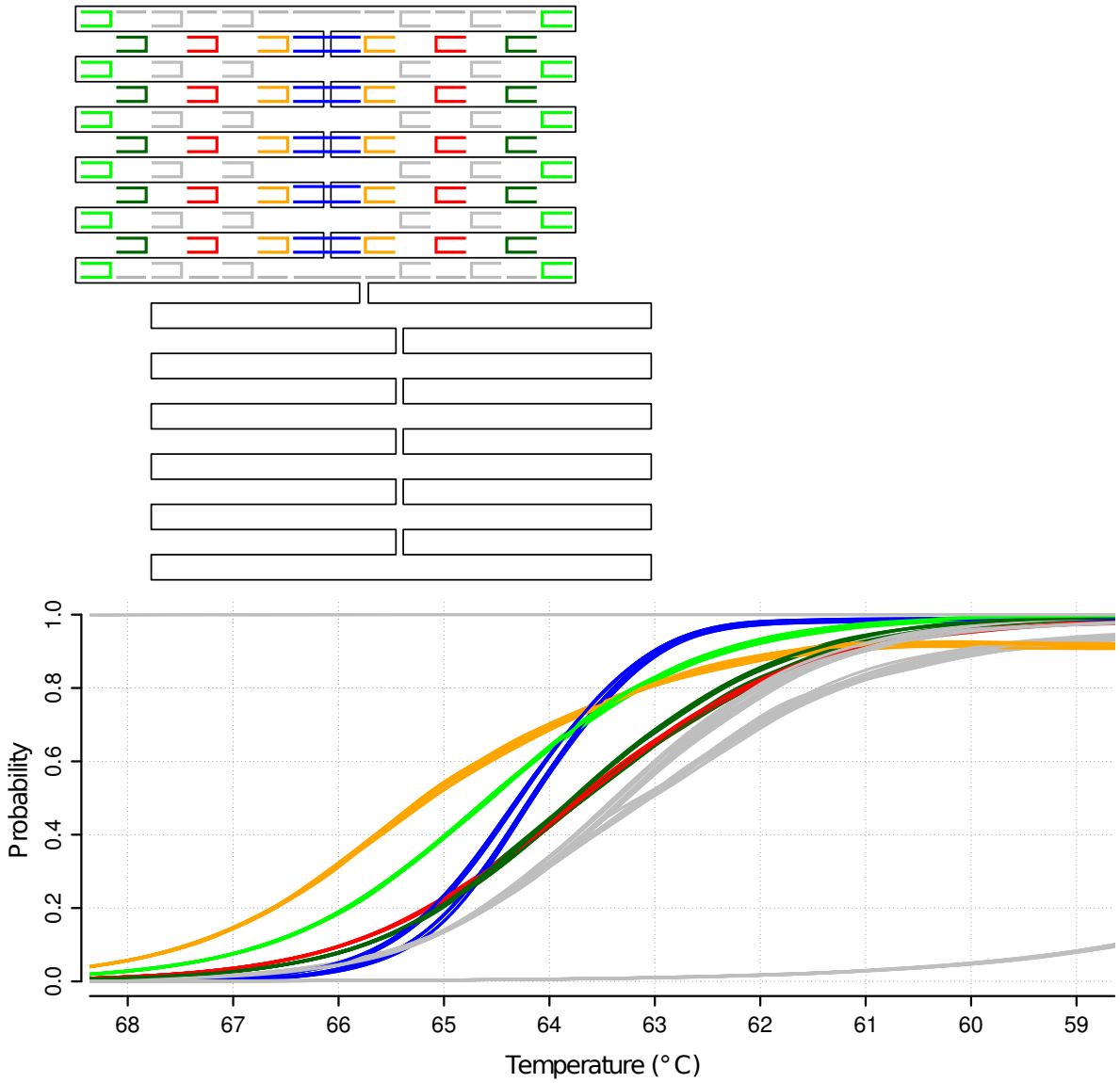


Figure 7.9: The average (smoothed) incorporation of each staple as a function of temperature, where 100% completion indicates two staples of the corresponding type are bound to the scaffold. Staples are colour-coded by their position in the tile. Double length (32-nt) single-domain staples are incorporated before the 68 °C mark (top grey line), while shorter (16-nt) single-domain staple hybridize to the scaffold at temperatures typically lower than 59 °C (bottom grey line).

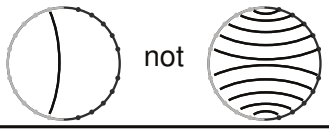
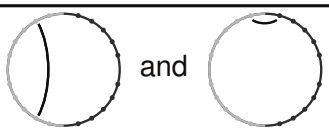
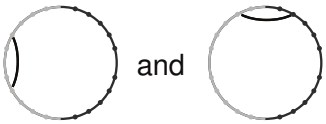
	5:0	4:1	3:2	N.P.	misfold	all
 not	117	8	5	5	82	217
 and	16	100	28	12	142	298
 and	20	19	155	11	176	381
Count:	**231**	**207**	**268**	**65**	**829**	**1600**

Figure 7.10: Adapted from [10]. The frequency of observing one of three properties (top to bottom) at $T = 64.21\,^{\circ}\mathrm{C}$ for a run of 1600 paths, subdivided by eventual fold. Approximately half the runs misfold (829), while the folds 5:0, 4:1 and 3:2 are observed roughly equally (231, 207 and 268 respectively). The N.P. folds are predicted less frequently (65×). The early behaviour of seam staples is highly indicative of the eventual fold. The occurrence of a seam connection that joins domains that are spaced 140 domains apart along the scaffold and no other seam connections joining in the opposite orientation is most likely to result in a 5:0 fold. The occurrence of a seam connection that joins domains that are spaced 112 domains apart along the scaffold and one seam connection occurring together with a seam connection in opposite orientation, while joining domains spaced 14 domains apart along the scaffold, is indicative of an eventual 4:1 fold. The occurrence of two seam connections in opposite orientation, each joining domains that are spaced 56-domains apart along the scaffold, is indicative of an eventual 3:2 fold.

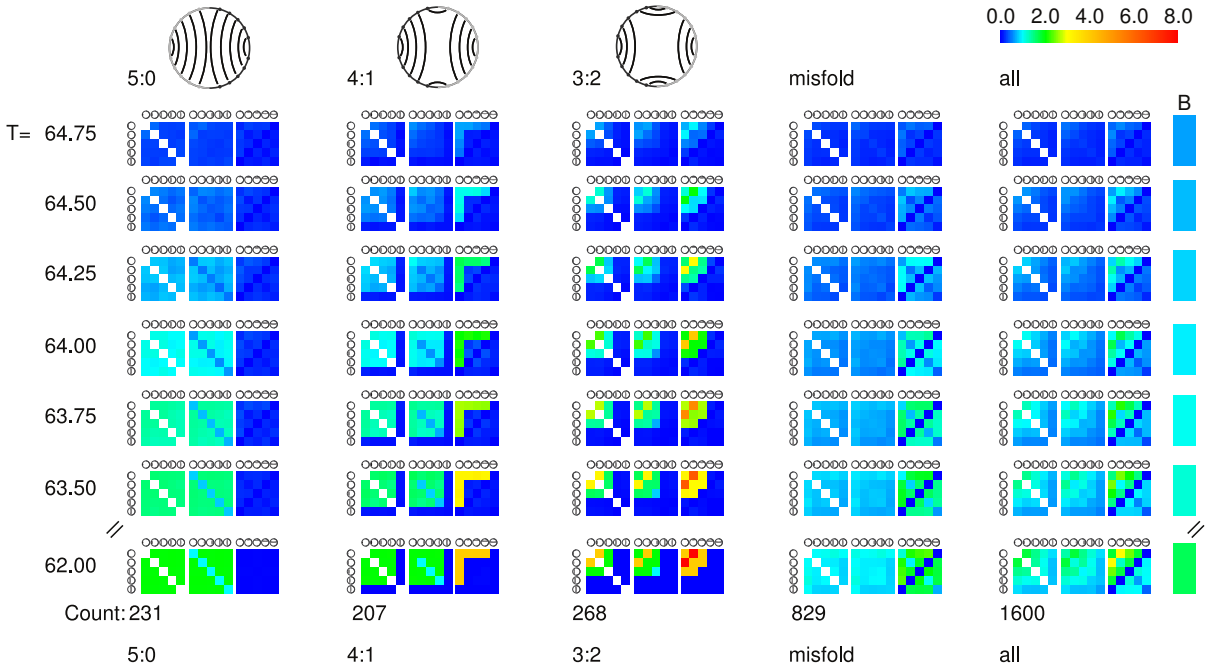


Figure 7.11: Adapted from [10]. Correlations between seam staples during annealing. Average data from 1600 simulations are presented ('all') together with subsets sorted by final fold (5:0, 4:1, 3:2 and misfold). Simulations resulting in well-folded, non-planar structures (NP) are included in 'all' but not presented separately: such structures occurred 65 times. Circular icons with internal connections of different lengths represent links across the seam ('seam links') connecting points on the scaffold spanning (i.e., that are separated by) 28, 56, 84, 112 and 140 scaffold domains. A 'seam link' represents a connection across the seam mediated by at least one seam staple. The rectangle on the right of the figure, labelled 'B', represents the average occupancy for each body-staple domain (range 0-2). Correlations between seam links are represented by three 5×5 blocks. Each pixel is labelled by two icons whose orientation has the same significance as in the 'circle' diagrams of Fig. 3: two icons related by 180° rotation represent one internal link in each of the two halves of the scaffold; a 90° rotation represents one internal link and one cross link. Each pixel represents the average number of pairs of links with the specified spans and relative orientations (range 0-8).

7.3.2 Broken seam staples

The cooperative pairing of seam staples is destroyed in the broken seam modification. One of each pair of seam staples is removed from the design, and replaced by two single domain staples, as in Fig. 7.13. The 50% incorporation mark of the remaining seam staples occurs 2.0 °C later on average due to the modification, and is lower than the average 50% incorporation mark of each body staple (see Fig. 7.13). The number of re-orientation events is drastically increased compared to the original design (Fig. 7.7ab). This modification results in the formation of 5:0 near exclusively, both in prediction and experiment (cf. Fig. 7.6b).

We provide an idealized description of the folding pathway. The relatively early incorporation of body staples results in an intermediate shape that can only result in the formation of 5:0 when the remaining seam staples are incorporated, see Fig. 7.12. The intermediate shape is attained frequently because it minimizes the loop distance for each of the body staples.

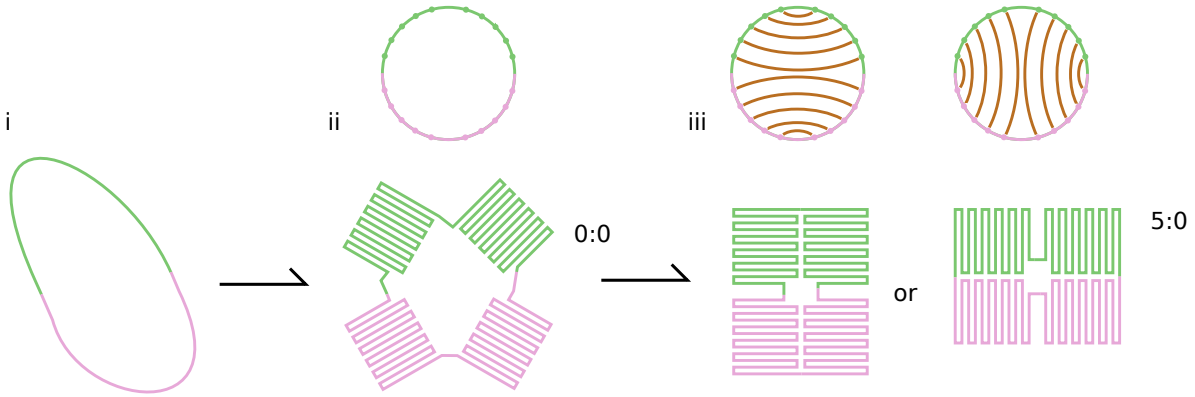


Figure 7.12: Idealized folding pathway for the broken seam modification. i) At high temperature, the scaffold is completely unhybridized. ii) Without the presence of seam staples, body staples bind in configurations that minimize the loop cost. iii) Adding the remaining seam staples results in the 5:0 fold.

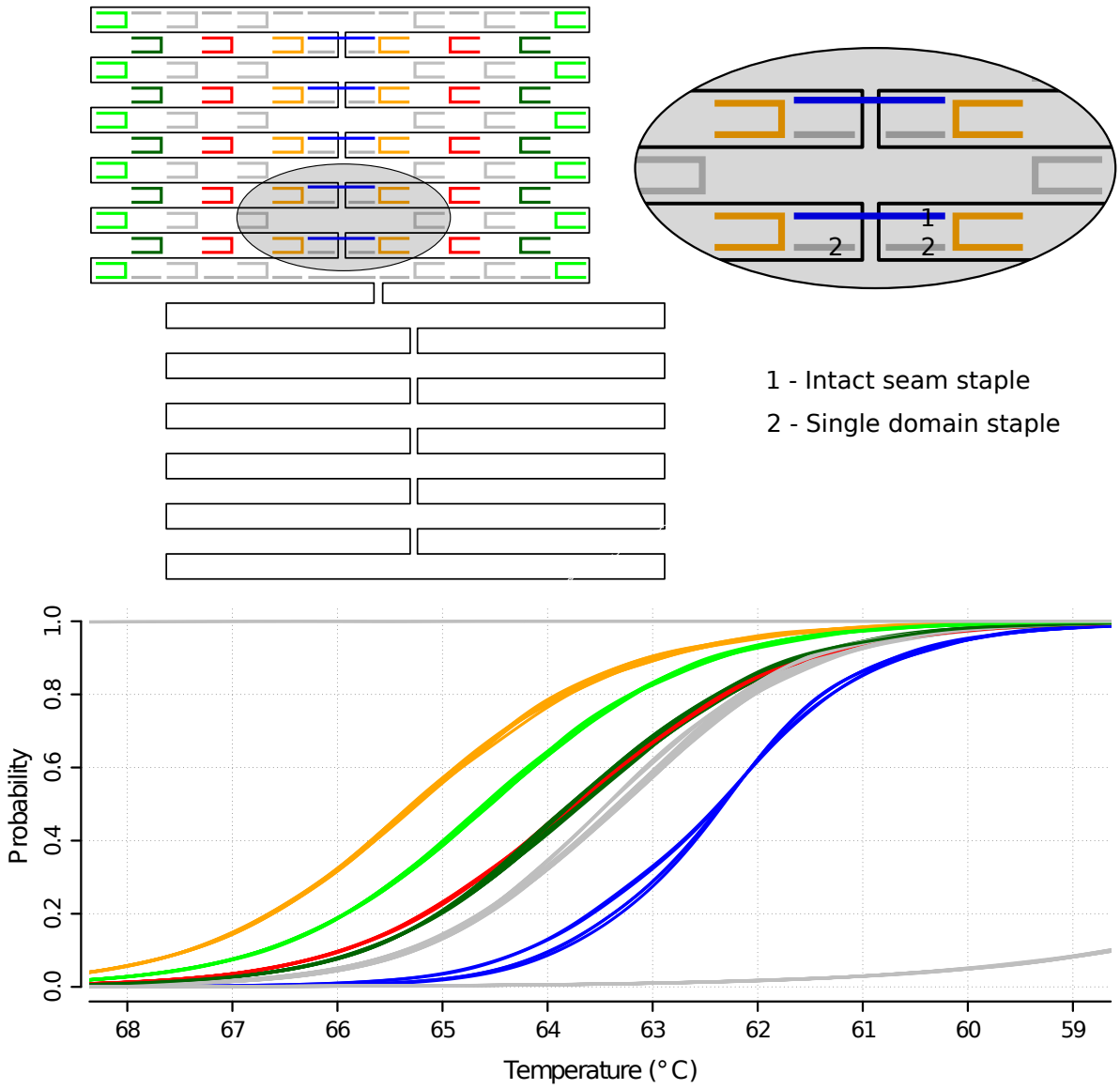


Figure 7.13: Top: The broken seam modification replaces every second seam staple with two 16-nt single-domain staples. Bottom: The average (smoothed) incorporation of each staple as a function of temperature for the broken seam modification. The remaining seam staples are significantly destabilized when compared to the original design (shown in Fig. 7.9).

7.3.3 Elongated staples

In the elongated staple modification, sixteen body staple domains are doubled in length, from 16 to 32 nucleotides as in Fig. 7.14, ensuring that these body staples hybridize to the scaffold well in advance of the remaining two-domain staples. In addition, the cooperativity within the middle pair of seam staples is destroyed, by replacing one of staples with two single-domain staples, similar to the broken seam modification. This modification typically leads to 5:0 or 3:2 folds both in prediction and experiment (cf. Fig. 7.6c). The annealing curves of the staples are affected as expected: the seam staple from the broken pair is incorporated later and the elongated staples are incorporated at higher temperatures than in the unmodified design (Fig. 7.14).

An idealized description of the pathway is as follows and we provide an illustration in Fig. 7.15. The elongated staples hybridize to the scaffold ahead of the other staples, and drastically reduce the distance between domains that are complementary to seam staples. The elongated staples and the remaining paired seam staples are equally unlikely to re-orientate during the folding, while the other staples exhibit increased re-orientation behaviour (Fig. 7.7ac). Incorporation of the remaining seam pairs is subsequently directed into the intermediate folds 2:2 or 4:0, necessarily suppressing the formation of the 4:1 shape.

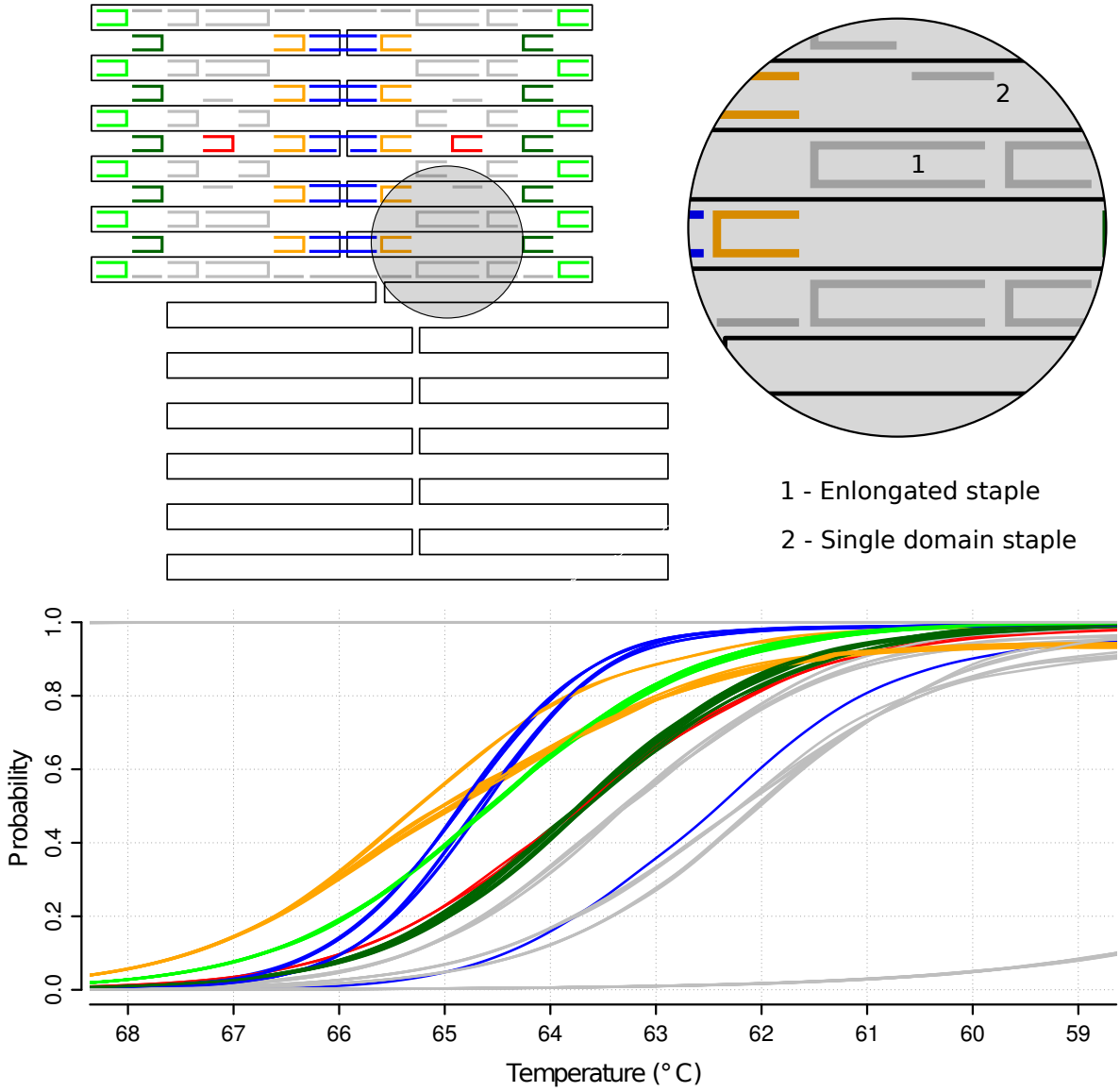


Figure 7.14: Top: The elongated staple modification doubles the length of the domains of eight body staples (in grey). To make room, necessarily eight staples are removed from the design (red, cf. Fig. 7.9) replaced with four 16-nt single-domain staples (in grey). The cooperativity of the middle-most seam pair is destroyed by replacing one of the pair with single-domain staples, similarly to the broken seam modification. Bottom: The average (smoothed) incorporation of each staple as function of temperature for the modification. The elongated staples have 32-nt domains and are incorporated fully by the 68 °C mark, and typically prevent the origami from reaching 4:1 folds.

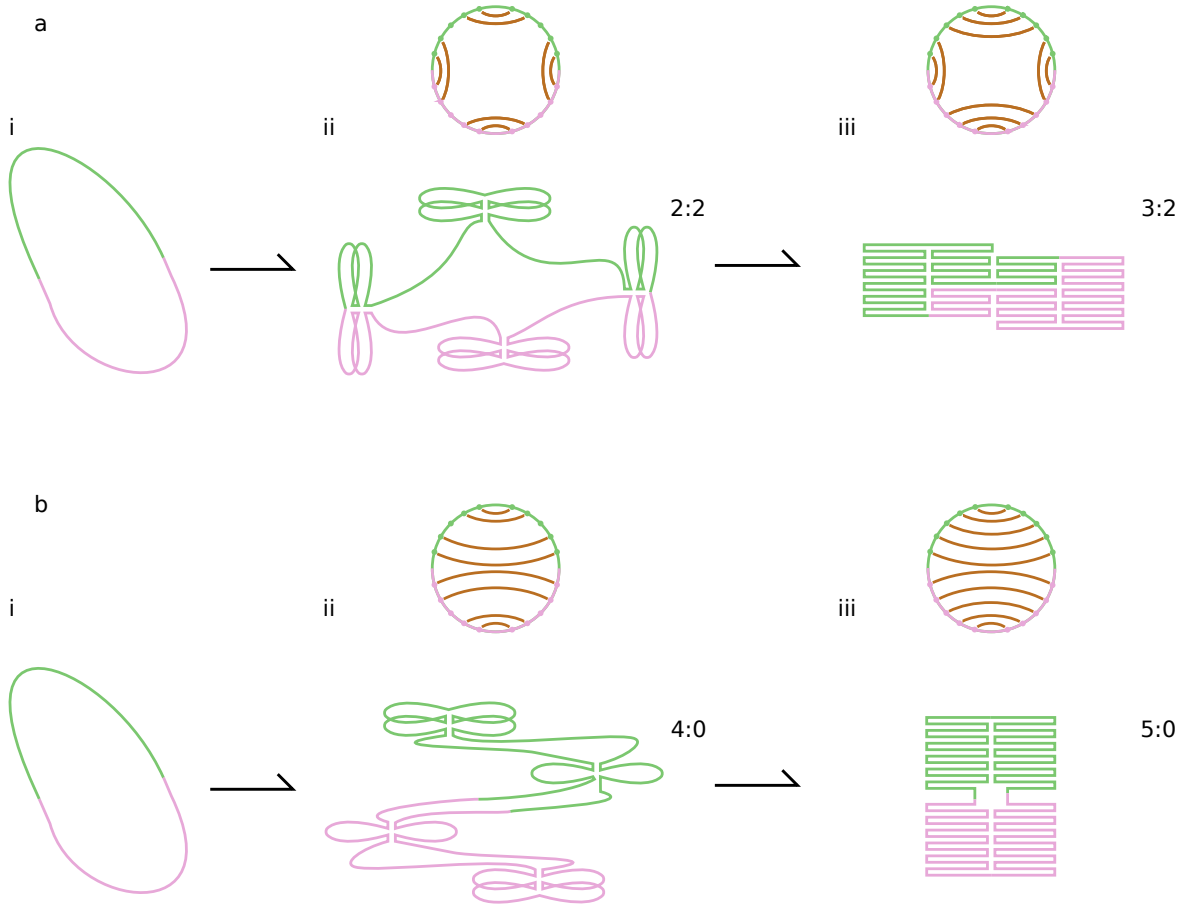


Figure 7.15: Idealized folding pathway for the elongated staples modification leading to shape 3:2 (a) or shape 5:0 (b). i) At high temperature, the scaffold is completely unhybridized. ii) The body staples with elongated (32-nt) domains bind to the scaffold well ahead of the other two-domain staples. The seam staples bind to the scaffold in one of two configurations (aii or bii). iii) The behaviour of the seam staples typically prevents the formation of the 4:1 shape.

7.3.4 Further modifications

The broken seam and the elongated staple modification did not drastically change the relative stability of the well-formed states, but did have a significant effect on the folding pathway. We discuss two further modifications, the alternative seam modification and the lower-right modification, which change both the folding pathway and the relative stability of the well-formed folds.

Alternative seam

The alternative seam is a further modification of the broken seam design, obtained by introducing one pair of staples on the edge of the tile. In Fig. 7.16a the modified tile is depicted in a 3:2 shape, where staples modified from the original design are depicted in black rather than grey. The steric constraints of the newly added pair of staples are not met by any well-formed states except that depicted in Fig. 7.16a. For example, observe the scaffold routing that results in a 4:1 shape in Fig. 7.16bi, where the newly added seam staples are depicted in red. In this fold the domains for the added staples do not align, and this state is excluded from the model. Without the newly added paired staples, the state would be legal, as in Fig. 7.16bii, and would count as a 4:1 tile in the model.

The added pair of staples are highly cooperative, and effective in shifting the pathway. The modification results in an increase of 3:2 shape (model) or shapes that are close to the expected offset of a 3:2 tile (experiment), as shown in Fig. 7.6d.

Lower-right

The lower-right modification places a 64-nt staple in the corner of the design. In Fig. 7.16c and 7.16d the modified tile is depicted in the 5:0*iv* and 4:1 shape. In all states other than the depicted 5:0*iv* shape, the long staple necessarily bends along with the scaffold, as in Fig. 7.16d. Double-stranded DNA is known to bend sharply in a 'kink', somewhat similar to a drinking straw that kinks when bent. The energy associated with kinking double-stranded DNA remains a topic of debate amongst researchers [169, 170]. In our model, we allow the kinked states without any additional penalty for the kinking itself, although loops that include the extra long domain are now destabilized additionally when compared to the unmodified design. In both the model and experiment the 5:0*iv* shape occurs at increased frequency (Fig. 7.6e).

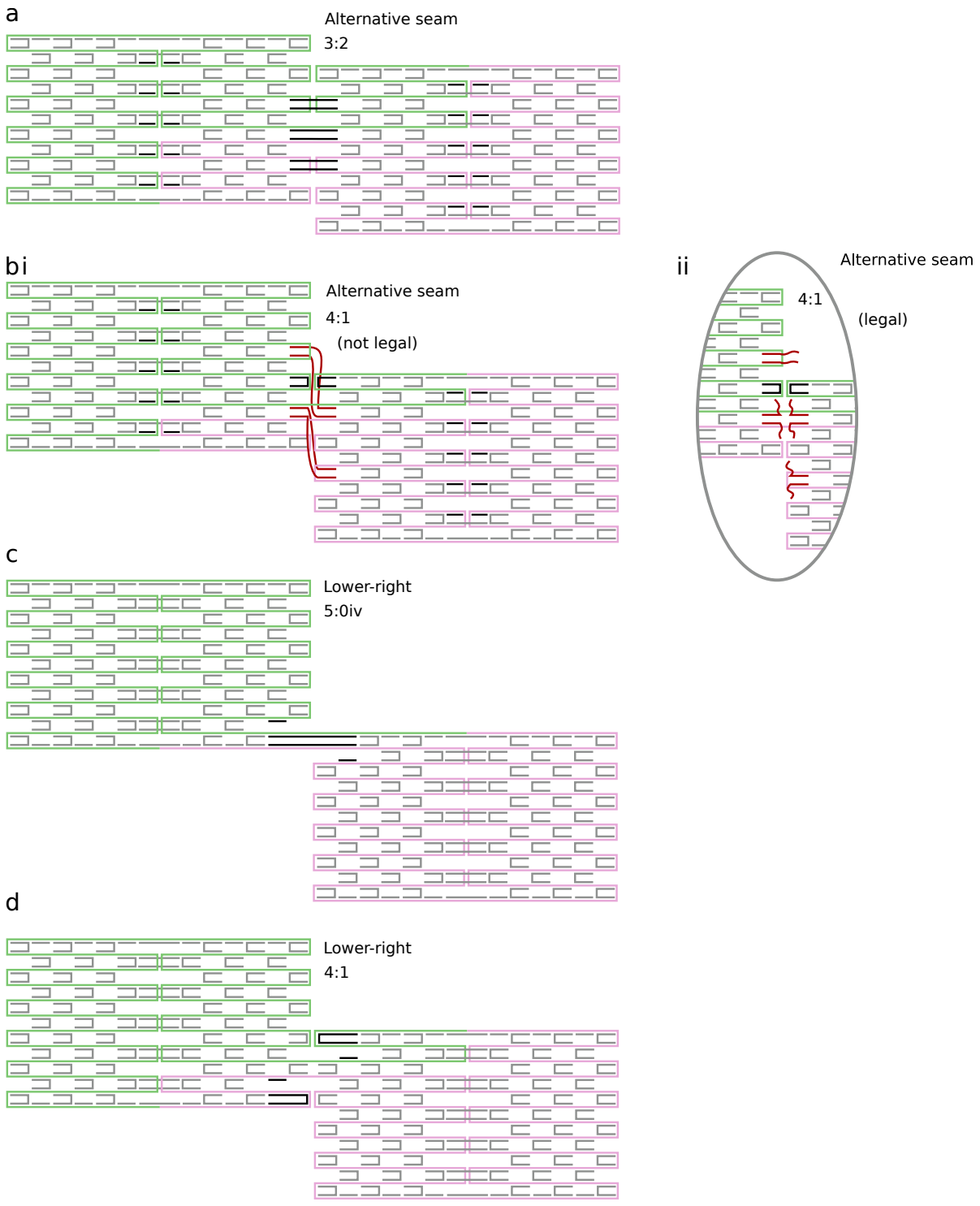


Figure 7.16: Further modifications to the polymorphic tile design. In black: the staples that differ from the original design. a) The alternative seam, depicted in the 3:2 fold. bi) The alternative seam, depicted in a 4:1 fold, excluded from the model. bii) The scaffold routing of (bi) is allowed when the crossover joins from the added seam (in red) are removed. c) The lower-right modification in a 5:0*iv* fold. d) The lower-right modification in the 4:1 shape.

7.4 Conclusion

In this chapter we applied the self-assembly model of Chapter 6 to a novel polymorphic tile. The polymorphic tile has multiple well-folded states that are distinguishable by shape. We have found the tile to assemble with high yield and the distribution of shapes to be highly programmable: this demonstrates that the assembly follows a well-defined pathway and we infer that a similar process occurs for regular origami. The model and experiment confirm that assembly is highly cooperative [153, 154] and highly sensitive to staple domain and crossover design [153, 155, 156]. The model predicts that, at early stages of folding, the reversible nature of domain hybridization is helpful to recover from kinetic traps.

In particular, the folding pathway of our polymorphic tile was found to be dominated by paired staple interactions, which the model predicts to be strongly cooperative. The preference to bind to the nearest binding site is a defining aspect of the model and was exploited effectively to steer the assembly pathway.

Chapter 8

Conclusions

Summary

The aim of this work was to develop models and model checking methods to increase the reliability of molecular devices made from DNA, specifically to benefit future application. Two applications were selected, that of DNA origami, a widely used technique to create molecular structures from DNA, and that of the molecular walker created by Wickham et al., a computational device that is embedded on an origami tile.

For the DNA walker we developed a model to describe the stepping of the walker and fitted a failure rate for the blocking mechanism. Our model identified modes of failure for the walker in the form of deadlock and unintended movement between tracks. We characterized the computational expressiveness of walker circuits of Wickham et al. and found the performance of the walker to be competitive compared to the performance of non-local strand displacement systems. We demonstrated, based on model predictions, how a DNA walker circuit can be optimized for performance ahead of implementation.

In the case of DNA origami we developed a thermodynamic assembly model by interpreting the partially folded structures as graphs. The model identifies looping constraints within the partially folded structure, and takes into account the thermodynamic cost of loop formation. The model is tunable over two parameters and reproduces the correct melting temperature and hysteresis observed in experiment. Further, it indicates that domain sequence, placement of crossovers and temperature gradient during assembly all affect the assembly pathway of DNA origami. Based on our model we control the folding pathway of a polymorphic DNA origami via seemingly minor modifications in the design.

We also developed a method for computing cumulative rewards of CTMC models based on fast adaptive uniformisation and applied it to DNA walker models, resulting in improved performance over standard uniformisation. We further developed a parameter synthesis method

for continuous-time Markov chains that is applicable when model parameters are uncertain. The method determines for which parameters the model satisfies a specified performance and/or reliability property and finds the rate that optimises the satisfaction of the property. We have also applied it to DNA walkers, synthesising rates that guarantee a given level of reliability.

Evaluation and future work

Probabilistic model checking We have attempted to apply probabilistic model checking to the rapidly evolving field of DNA nanotechnology, which encompasses many devices and techniques. The studies of the DNA walker indicate that exhaustive model checking of continuous-time models that simulate mechanistic processes is troublesome both for explicit and symbolic methods. The fast adaptive uniformisation method performs an order of magnitude better than standard uniformisation, but because of the exponential blow-up this does not translate into meaningful gains and future endeavours should rely on the use of simulation, perhaps in conjunction with numerical methods. The application of parameter synthesis to DNA nanotechnology seems similarly premature: a prediction of which parameters are required to meet a predetermined performance does not tell us *how* the device should be modified to achieve such performance. Within this context an interesting question is: is it possible to algorithmically synthesize walker circuit layouts with optimal performance, or perhaps to synthesize walker circuit layouts that meet a pre-determined performance level? The application of automated design algorithms in microprocessor manufacturing could be relevant here.

On the upside, statistical model checking was demonstrated to be a highly practical method to analyse system-wide performance of the walker circuits, an approach that was subsequently investigated in [171]. Formal verification may still be useful to verify the correct design and logic of reaction networks or localized DNA computers, but improvements in scalability are needed.

Robust DNA computers and devices We have also attempted to provide high-level models of the DNA walker. To create better models of the walker, more calibration data would be highly beneficial. Unfortunately, also fundamental understanding of certain DNA processes is currently lacking. Crucially, DNA strand displacement (Section 2.3) is widely used, but we only generally know the dependence of reaction kinetics on buffer conditions, temperature and nucleotide sequence. More knowledge of this process is required if we were to reliably predict the performance of the current generation of DNA computers in real-world circumstances. Furthermore, detailed knowledge of the mechanics of strand displacement might directly inspire new applications. Recent work on this topic includes [82, 83, 84, 85].

Secondly, the domain-level abstraction is leaky within the context of DNA strand displacement networks, where partial domain overlap is believed to cause leakage reactions [11], but

their workings is not well known. To improve DNA strand displacement networks, a better understanding of these leakage rates is required. The mechanics and speed of blunt-end strand displacement are similarly not well characterized (that is, kinetics are known to about an order of magnitude, independent from sequence) at time of writing.

DNA self-assembly The discrete stochastic model developed in Chapter 6 was highly successful in predicting the eventual shape of the polymorphic tile of Chapter 7. However, to simulate the polymorphic tile we assumed that every domain was equally stable, which may not be justified. We also employed the self-assembly model at model parameters that in Chapter 6 were demonstrated to show too little hysteresis ($\gamma = 1.5, n = 0$). Better calibration of the model is required to resolve these discrepancies. Additional terms to take into account the thermodynamic cost of duplex bending may be added in future.

The lack of support for steric constraints is perhaps the largest inaccuracy in our simulation and it remains to be seen if a representation different from molecular dynamics could mitigate this drawback: to find one might pose a challenge. The problem is as follows: given a set of interconnected cylinders (DNA helices) and their volume, is there a fast, possibly approximate, method to decide if a non-overlapping configuration exists? The exclusion algorithm worked well for the simulation of the polymorphic tile, but is a relatively coarse approximation to the steric constraints that, in the current form, does not apply to regular origami. An efficient description of the steric constraints may enable models to reproduce the poor yield of multi-layered origami [153] or accurately simulate the influence of staple design [155]. To better understand the assembly of DNA origami, and to enable prediction of yield and more accurate assembly kinetics, it is likely that (scalable) coarse grained models are required.

We demonstrated a DNA origami structure that intentionally incorporates multiple copies of the same staple during the assembly process. Further research, aimed to design functional DNA nano-structures while re-using strands at multiple points in the design, may decrease the direct materials cost and simplify the logistics of nanostructure manufacturing. Direct design of the folding pathways may then be employed to increase yield of the intended structure, and to identify pathways that lead to misfolds. Detailed simulation of folding pathways may help to create complex devices that assemble at constant temperature.

In the future, complex and functional nanostructures will be constructed by *other* nanodevices, that chaperone the device through various stages of production. Chaperoning devices may shield parts of the structure from alteration, or organize the reactants through spatial separation, or catalyse the assembly between components directly. Similarly, the activation and control of artificial nano-machinery will occur through *recognition domains*, where the docking of one molecule activates the device, enabling interaction between physically separated nanodevices.

References

[1] F. Dannenberg, "DNA Walkers PRISM case study." `http://www.prismmodelchecker.org/casestudies/dna_walkers.php` and `http://www.prismmodelchecker.org/files/dna19/dna_walkers_benchmark.zip`. Accessed: 2015-05-12.

[2] F. Dannenberg, "Self-assembly simulation code." `https://github.com/fdannenberg/dna`. Accessed: 2015-05-12.

[3] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. J. Turberfield, "DNA walker circuits: Computational potential, design, and verification," in *DNA Computing and Molecular Programming* (D. Soloveichik and B. Yurke, eds.), vol. 8141 of *Lecture Notes in Computer Science*, pp. 31–45, Springer International Publishing, 2013.

[4] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. Turberfield, "DNA walker circuits: computational potential, design, and verification," *Natural Computing*, vol. 14, pp. 195–211, 2015.

[5] F. Dannenberg, E. M. Hahn, and M. Kwiatkowska, "Computing cumulative rewards using fast adaptive uniformisation," in *Computational Methods in Systems Biology*, pp. 33–49, Springer, 2013.

[6] F. Dannenberg, E. M. Hahn, and M. Kwiatkowska, "Computing cumulative rewards using fast adaptive uniformization," *ACM Transactions on Modeling and Computer Simulation*, vol. 25, pp. 9:1–9:23, 2015.

[7] M. Češka, F. Dannenberg, M. Kwiatkowska, and N. Paoletti, "Precise parameter synthesis for stochastic biochemical systems," in *Computational Methods in Systems Biology* (P. Mendes, J. Dada, and K. Smallbone, eds.), vol. 8859 of *Lecture Notes in Computer Science*, pp. 86–98, Springer International Publishing, 2014.

[8] M. Češka, F. Dannenberg, M. Kwiatkowska, and N. Paoletti, "Precise parameter synthesis for stochastic biochemical systems," in *Journal version, submitted.*

[9] F. Dannenberg, K. E. Dunn, J. Bath, M. Kwiatkowska, A. J. Turberfield, and T. E. Ouldridge, "Modelling DNA origami self-assembly at the domain level," *Under review.*

[10] K. E. Dunn, F. Dannenberg, T. E. Ouldridge, M. Kwiatkowska, A. J. Turberfield, and J. Bath, "Guiding the folding pathway of DNA origami," *Nature*, pp. 82–86, 2015.

[11] L. Qian and E. Winfree, "Scaling up digital circuit computation with DNA strand displacement cascades," *Science*, vol. 332, pp. 1196–201, 2011.

[12] Y.-J. Chen, N. Dalchau, N. Srinivas, A. Phillips, L. Cardelli, D. Solveichik, and G. Seelig, "Programmable chemical controllers made from DNA," *Nature Nanotechnology*, vol. 8, pp. 755–762, 2013.

[13] N. a. W. Bell, C. R. Engst, M. Ablay, G. Divitini, C. Ducati, T. Liedl, and U. F. Keyser, "DNA origami nanopores.," *Nano letters*, vol. 12, pp. 512–7, 2012.

[14] M. Langecker, V. Arnaut, T. G. Martin, J. List, S. Renner, M. Mayer, H. Dietz, and F. C. Simmel, "Synthetic lipid membrane channels formed by designed DNA nanostructures," *Science*, vol. 338, pp. 932–936, 2012.

[15] R. Carlson, "The changing economics of DNA synthesis," *Nature biotechnology*, vol. 27, no. 12, p. 1091, 2009.

[16] L. Qian and E. Winfree, "A simple DNA gate motif for synthesizing large-scale circuits," in *DNA Computing* (A. Goel, F. Simmel, and P. Sosk, eds.), vol. 5347 of *Lecture Notes in Computer Science*, pp. 70–89, Springer Berlin Heidelberg, 2009.

[17] L. Qian, E. Winfree, and J. Bruck, "Neural network computation with DNA strand displacement cascades," *Nature*, vol. 475, no. 7356, pp. 368–72, 2011.

[18] B. Yurke, A. J. Turberfield, A. P. Mills, F. C. Simmel, and J. L. Neumann, "A DNA-fuelled molecular machine made of DNA," *Nature*, vol. 406, pp. 605–608, 2000.

[19] P. W. K. Rothemund, "Folding DNA to create nanoscale shapes and patterns," *Nature*, vol. 440, pp. 297–302, 2006.

[20] H. G. Hansma and D. E. Laney, "DNA binding to mica correlates with cationic radius: assay by atomic force microscopy.," *Biophysical journal*, vol. 70, p. 1933, 1996.

[21] Y. Ke, S. M. Douglas, M. Liu, J. Sharma, A. Cheng, A. Leung, Y. Liu, W. M. Shih, and H. Yan, "Multilayer DNA origami packed on a square lattice," *Journal of the American Chemical Society*, vol. 131, pp. 15903–15908, 2009.

[22] X.-C. Bai, T. G. Martin, S. H. W. Scheres, and H. Dietz, "Cryo-EM structure of a 3D DNA-origami object," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 109, pp. 20012–20017, 2012.

[23] L. E. Morrison and L. M. Stols, "Sensitive fluorescence-based thermodynamic and kinetic measurements of DNA hybridization in solution," *Biochemistry*, vol. 32, pp. 3095–104, 1993.

[24] J. SantaLucia, "A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 95, pp. 1460–1465, 1998.

[25] A. Panjkovich and F. Melo, "Comparison of different melting temperature calculation methods for short DNA sequences," *Bioinformatics*, vol. 21, pp. 711–722, 2005.

[26] J. SantaLucia and D. Hicks, "The thermodynamics of DNA structural motifs," *Annual Review of Biophysics and Biomolecular Structure*, vol. 33, pp. 415–40, 2004.

[27] S. Bommarito, N. Peyret, and J. SantaLucia, "Thermodynamic parameters for DNA sequences with dangling ends," *Nucleic Acids Research*, vol. 28, pp. 1929–1934, 2000.

[28] N. C. Seeman, "Nucleic acid junctions and lattices," *Journal of Theoretical Biology*, vol. 99, pp. 237–247, 1982.

[29] N. R. Kallenbach, R. I. Ma, and N. C. Seeman, "An immobile nucleic acid junction constructed from oligonucleotides," *Nature*, vol. 305, pp. 829–831, 1983.

[30] J. Chen and N. C. Seeman, "Synthesis from DNA of a molecule with the connectivity of a cube," *Nature*, vol. 350, pp. 631–633, 1991.

[31] Y. Zhang and N. Seeman, "Construction of a DNA-truncated octahedron," *Journal of American Chemical Society*, vol. 116, pp. 1661–1669, 1994.

[32] R. P. Goodman, I. A. T. Schaap, C. F. Tardin, C. M. Erben, R. M. Berry, C. F. Schmidt, and A. J. Turberfield, "Rapid chiral assembly of rigid DNA building blocks for molecular nanofabrication," *Science*, vol. 310, pp. 1661–1665, 2005.

[33] C. M. Erben, R. P. Goodman, and A. J. Turberfield, "A self-assembled DNA bipyramid.," *Journal of the American Chemical Society*, vol. 129, pp. 6992–3, 2007.

[34] E. S. Andersen, M. Dong, M. M. Nielsen, K. Jahn, R. Subramani, W. Mamdouh, M. M. Golas, B. Sander, H. Stark, C. L. P. Oliveira, J. S. Pedersen, V. Birkedal, F. Besenbacher,

K. V. Gothelf, and J. Kjems, "Self-assembly of a nanoscale DNA box with a controllable lid," *Nature*, vol. 459, pp. 73–76, 2009.

[35] S. M. Douglas, I. Bachelet, and G. M. Church, "A logic-gated nanorobot for targeted transport of molecular payloads," *Science*, vol. 335, pp. 831–4, 2012.

[36] S. Woo and P. W. K. Rothemund, "Programmable molecular recognition based on the geometry of DNA nanostructures.," *Nature Chemistry*, vol. 3, pp. 620–7, 2011.

[37] T. Gerling, K. F. Wagenbauer, A. M. Neuner, and H. Dietz, "Dynamic DNA devices and assemblies formed by shape-complementary, non–base pairing 3D components," *Science*, vol. 347, pp. 1446–1452, 2015.

[38] B. Wei, M. Dai, and P. Yin, "Complex shapes self-assembled from single-stranded DNA tiles.," *Nature*, vol. 485, pp. 623–6, 2012.

[39] S. M. Douglas, A. H. Marblestone, S. Teerapittayanon, A. Vazquez, G. M. Church, and W. M. Shih, "Rapid prototyping of 3D DNA-origami shapes with caDNAno," *Nucleic Acids Research*, vol. 37, pp. 5001–6, 2009.

[40] D.-N. Kim, F. Kilchherr, H. Dietz, and M. Bathe, "Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures," *Nucleic Acids Research*, vol. 40, pp. 2862–2868, 2012.

[41] W. B. Sherman, , and N. C. Seeman, "A precisely controlled DNA biped walking device," *Nano Letters*, vol. 4, pp. 1203–1207, 2004.

[42] J. S. Shin and N. A. Pierce, "A synthetic DNA walker for molecular transport," *Journal of the American Chemical Society*, vol. 126, pp. 10834–5, Sept. 2004.

[43] J. Bath, S. J. Green, and A. J. Turberfield, "A free-running DNA motor powered by a nicking enzyme," *Angewandte Chemie International Edition*, vol. 44, pp. 4358–4361, 2005.

[44] S. F. J. Wickham, M. Endo, Y. Katsuda, K. Hidaka, J. Bath, H. Sugiyama, and A. J. Turberfield, "Direct observation of stepwise movement of a synthetic molecular transporter," *Nature Nanotechnology*, vol. 6, pp. 166–169, 2011.

[45] S. F. J. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. J. Turberfield, "A DNA-based molecular motor that can navigate a network of tracks," *Nature Nanotechnology*, vol. 7, pp. 169–173, 2012.

[46] P. Yin, H. Yan, X. G. Daniell, A. J. Turberfield, and J. H. Reif, "A unidirectional DNA walker that moves autonomously along a track," *Angewandte Chemie International Edition*, vol. 43, pp. 4906–4911, 2004.

[47] S. Green, J. Bath, and A. Turberfield, "Coordinated chemomechanical cycles: a mechanism for autonomous molecular motion," *Physical Review Letters*, vol. 101, p. 238101, 2008.

[48] J. Bath, S. J. Green, K. E. Allen, and A. J. Turberfield, "Mechanism for a directional, processive, and reversible DNA motor," *Small*, vol. 5, pp. 1513–1516, 2009.

[49] T. Omabegho, R. Sha, and N. C. Seeman, "A bipedal DNA Brownian motor with coordinated legs," *Science*, vol. 324, pp. 67–71, 2009.

[50] T.-G. Cha, J. Pan, H. Chen, J. Salgado, X. Li, C. Mao, and J. H. Choi, "A synthetic DNA motor that transports nanoparticles along carbon nanotubes," *Nature Nanotechnology*, vol. 9, pp. 39–43, 2014.

[51] L. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, pp. 1021–1023, 1994.

[52] K. Sakamoto, "Molecular computation by DNA hairpin formation," *Science*, vol. 288, pp. 1223–1226, 2000.

[53] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. Adleman, "Solution of a 20-variable 3-SAT problem on a DNA computer," *Science*, vol. 296, pp. 499–502, 2002.

[54] H. Wang, "Proving theorems by pattern recognition-II," *Bell System Technical Journal*, vol. 40, pp. 1–41, 1961.

[55] R. Berger, *The undecidability of the domino problem*, vol. 66. Memoirs of the American Mathematical Society, 1966.

[56] T. J. Fu and N. C. Seeman, "DNA double-crossover molecules," *Biochemistry*, vol. 32, no. 13, pp. 3211–3220, 1993.

[57] E. Winfree, "On the computational power of DNA annealing and ligation," *DNA Based Computers. DIMACS Series in Discrete Mathematics and Theoretical Computer Science.*, vol. 27, pp. 99–221, 1996.

[58] E. Winfree, *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

[59] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman, "Design and self-assembly of two-dimensional DNA crystals," *Nature*, vol. 394, pp. 539–44, 1998.

[60] E. Winfree, "Simulations of computing by self-assembly," *Unpublished*, 1998.

[61] P. W. K. Rothemund, N. Papadakis, and E. Winfree, "Algorithmic self-assembly of DNA sierpinski triangles," *PLoS Biol*, vol. 2, p. 424, 2004.

[62] R. D. Barish, P. W. K. Rothemund, and E. Winfree, "Two computational primitives for algorithmic self-assembly: copying and counting," *Nano Letters*, vol. 5, pp. 2586–2592, 2005.

[63] D. Soloveichik, G. Seelig, and E. Winfree, "DNA as a universal substrate for chemical kinetics," *Proceedings of the National Academy of Sciences*, vol. 107, no. 12, pp. 5393–5398, 2010.

[64] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, "An autonomous molecular computer for logical control of gene expression," *Nature*, vol. 429, pp. 423–429, 2004.

[65] G. Seelig, D. Soloveichik, D. Y. Zhang, and E. Winfree, "Enzyme-free nucleic acid logic circuits," *Science*, vol. 314, no. 5805, pp. 1585–1588, 2006.

[66] D. Y. Zhang, A. J. Turberfield, B. Yurke, and E. Winfree, "Engineering entropy-driven reactions and networks catalyzed by DNA," *Science*, vol. 318, pp. 1121–5, 2007.

[67] B. Li, Y. Jiang, X. Chen, and A. D. Ellington, "Probing spatial organization of DNA strands using enzyme-free hairpin assembly circuits," *Journal of the American Chemical Society*, vol. 134, pp. 13918–13921, 2012.

[68] B. Li, A. D. Ellington, and X. Chen, "Rational, modular adaptation of enzyme-free DNA circuits to multiple detection methods," *Nucleic acids research*, vol. 39, p. 110, 2011.

[69] J. Milligan and A. Ellington, "Using RecA protein to enhance kinetic rates of DNA circuits," *Chemical Communications*, vol. 51, pp. 9503–9506, 2015.

[70] L. Cardelli, "Strand algebras for DNA computing," in *DNA Computing and Molecular Programming* (R. Deaton and A. Suyama, eds.), vol. 5877 of *Lecture Notes in Computer Science*, pp. 12–24, Springer Berlin Heidelberg, 2009.

[71] A. Phillips and L. Cardelli, "A programming language for composable DNA circuits," *Journal of the Royal Society Interface*, vol. 6, pp. 419–36, 2009.

[72] M. Lakin, S. Youssef, L. Cardelli, and A. Phillips, "Abstractions for DNA circuit design," *Journal of the Royal Society Interface*, vol. 9, pp. 470–486, 2012.

[73] L. Cardelli, "Two-domain DNA strand displacement," *Mathematical Structures in Computer Science*, vol. 23, pp. 247–271, 2013.

[74] C. Grun, K. Sarma, B. Wolfe, S. W. Shin, and E. Winfree, "A domain-level DNA strand displacement reaction enumerator allowing arbitrary non-pseudoknotted secondary structures," *ArXiv*, vol. abs/1505.03738, 2015.

[75] C. Thachuk and A. Condon, "Space and energy efficient computation with DNA strand displacement systems," *DNA Computing and Molecular Programming*, 2012.

[76] M. R. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips, "Design and analysis of DNA strand displacement devices using probabilistic model checking," *Journal of the Royal Society Interface*, vol. 9, pp. 1470–85, 2012.

[77] R. Petersen, M. R. Lakin, and A. Phillips, "A strand graph semantics for DNA-based computation," *Theoretical Computer Science*, 2015.

[78] N. Peyret, *Prediction of nucleic acid hybridization: parameters and algorithms.* PhD thesis, Wayne State University, 2000.

[79] E. Protozanova, P. Yakovchuk, and M. D. Frank-Kamenetskii, "Stacked unstacked equilibrium at the nick site of DNA," *Journal of Molecular Biology*, vol. 342, no. 3, pp. 775 – 785, 2004.

[80] M. Zuker, "Mfold web server for nucleic acid folding and hybridization prediction," *Nucleic Acids Research*, vol. 31, pp. 3406–3415, 2003.

[81] R. M. Dirks, J. S. Bois, J. M. Schaeffer, E. Winfree, and N. A. Pierce, "Thermodynamic analysis of interacting nucleic acid strands," *SIAM Review*, vol. 49, pp. 65–88, 2007.

[82] D. Y. Zhang and E. Winfree, "Control of DNA strand displacement kinetics using toehold exchange," *Journal of the American Chemical Society*, vol. 131, pp. 17303–17314, 2009.

[83] T. E. Ouldridge, P. Šulc, F. Romano, J. P. Doye, and A. A. Louis, "DNA hybridization kinetics: zippering, internal displacement and sequence dependence," *Nucleic acids research*, vol. 41, pp. 8886–8895, 2013.

[84] N. Srinivas, T. E. Ouldridge, P. Šulc, J. M. Schaeffer, B. Yurke, A. A. Louis, J. P. Doye, and E. Winfree, "On the biophysics and kinetics of toehold-mediated DNA strand displacement," *Nucleic Acids Research*, vol. 41, pp. 10641–10658, 2013.

[85] R. R. Machinek, T. E. Ouldridge, N. E. Haley, J. Bath, and A. J. Turberfield, "Programmable energy landscapes for kinetic control of DNA strand displacement," *Nature Communications*, vol. 5, 2014.

[86] T. E. Ouldridge, A. A. Louis, and J. P. K. Doye, "Structural, mechanical, and thermodynamic properties of a coarse-grained DNA model.," *Journal of Chemical Physics*, vol. 134, Feb. 2011.

[87] T. E. Ouldridge, I. G. Johnston, A. A. Louis, and J. P. K. Doye, "The self-assembly of DNA Holliday junctions studied with a minimal model," *Journal of Chemical Physics*, vol. 130, 2009.

[88] T. E. Ouldridge, A. A. Louis, and J. P. K. Doye, "DNA nanotweezers studied with a coarse-grained model of DNA," *Physical Review Letters*, vol. 104, p. 178101, 2010.

[89] P. Šulc, T. E. Ouldridge, F. Romano, J. P. Doye, and A. A. Louis, "Simulating a burnt-bridges DNA motor with a coarse-grained DNA model," *Natural Computing*, vol. 13, pp. 535–547, 2014.

[90] J. Yoo and A. Aksimentiev, "In situ structure and dynamics of DNA origami determined through molecular dynamics simulations," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, pp. 20099–20104, 2013.

[91] B. Kuhlman, G. Dantas, G. C. Ireton, G. Varani, B. L. Stoddard, and D. Baker, "Design of a novel globular protein fold with atomic-level accuracy," *Science*, vol. 302, pp. 1364–1368, 2003.

[92] N. P. King, W. Sheffler, M. R. Sawaya, B. S. Vollmar, J. P. Sumida, I. Andr, T. Gonen, T. O. Yeates, and D. Baker, "Computational design of self-assembling protein nanomaterials with atomic level accuracy," *Science*, vol. 336, no. 6085, pp. 1171–1174, 2012.

[93] H. Gradišar, S. Božič, T. Doles, D. Vengust, I. Hafner-Bratkovič, A. Mertelj, B. Webb, A. Šali, S. Klavžar, and R. Jerala, "Design of a single-chain polypeptide tetrahedron assembled from coiled-coil segments," *Nature Chemical Biology*, vol. 9, pp. 362–366, 2013.

[94] T. Jiang, C. Xu, Y. Liu, Z. Liu, J. S. Wall, X. Zuo, T. Lian, K. Salaita, C. Ni, D. Pochan, *et al.*, "Structurally defined nanoscale sheets from self-assembly of collagen-mimetic peptides," *Journal of the American Chemical Society*, vol. 136, pp. 4300–4308, 2014.

[95] C. Geary, P. W. K. Rothemund, and E. S. Andersen, "A single-stranded architecture for cotranscriptional folding of RNA nanostructures," *Science*, vol. 345, pp. 799–804, 2014.

[96] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for Performance Evaluation* (M. Bernardo and J. Hillston, eds.), vol. 4486 of *Lecture Notes in Computer Science*, pp. 220–270, Springer Berlin Heidelberg, 2007.

[97] C. Baier, B. Haverkort, H. Hermanns, and J. Katoen, "Model-checking algorithms for continuous-time Markov chains," *IEEE Transactions on Software Engineering*, vol. 29, pp. 524–541, June 2003.

[98] A. Reibman, "Numerical transient analysis of Markov models," *Computers & Operations Research*, vol. 15, pp. 19–36, 1988.

[99] W. Grassmann, "Transient solutions in Markovian queues: an algorithm for finding them and determining their waiting-time distributions," *European Journal of Operational Research*, vol. 1, pp. 396 – 402, 1977.

[100] A. Jensen, "Markoff chains as an aid in the study of Markoff processes," *Scandinavian Actuarial Journal*, vol. 1953, pp. 87–91, 1953.

[101] B. L. Fox and P. W. Glynn, "Computing Poisson probabilities," *Communications of the ACM*, vol. 31, pp. 440–445, 1988.

[102] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *The Journal of Physical Chemistry*, vol. 81, pp. 2340–2361, 1977.

[103] T. Han, J.-P. Katoen, and A. Mereacre, "Compositional modeling and minimization of time-inhomogeneous markov chains," in *Hybrid Systems: Computation and Control* (M. Egerstedt and B. Mishra, eds.), vol. 4981 of *Lecture Notes in Computer Science*, pp. 244–258, Springer Berlin Heidelberg, 2008.

[104] N. M. van Dijk, "Uniformization for nonhomogeneous Markov chains," *Operations Research Letters*, vol. 12, pp. 283 – 291, 1992.

[105] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton, "Verifying continuous time Markov chains," in *Proceedings of the 8th International Conference on Computer Aided Verification*, pp. 269–276, Springer-Verlag, 1996.

[106] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proceedings of the 23rd International Conference on Computer Aided Verification*, pp. 585–591, Springer-Verlag, 2011.

[107] M. Kwiatkowska, G. Norman, and A. Pacheco, "Model checking expected time and expected reward formulae with random time bounds," *Computers & Mathematics with Applications*, vol. 51, pp. 305–316, 2006.

[108] A. Reibman and K. Trivedi, "Transient analysis of cumulative measures of Markov model behavior," *Stochastic Models*, vol. 5, pp. 683–710, 1989.

[109] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE Transactions on Software Engineering*, pp. 125–143, 1977.

[110] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," *Journal of the ACM*, vol. 32, pp. 733–749, 1985.

[111] E. A. Emerson and J. Y. Halpern, "Decision procedures and expressiveness in the temporal logic of branching time," *Journal of Computer and System Sciences*, vol. 30, pp. 1 – 24, 1985.

[112] R. Kaivola, R. Ghughal, N. Narasimhan, A. Telfer, J. Whittemore, S. Pandav, A. Slobodov, C. Taylor, V. Frolov, E. Reeber, and A. Naik, "Replacing testing with formal verification in Intel® core^{tm} i7 processor execution engine validation," in *Computer Aided Verification* (A. Bouajjani and O. Maler, eds.), vol. 5643 of *Lecture Notes in Computer Science*, pp. 414–429, Springer Berlin Heidelberg, 2009.

[113] T. Ball, B. Cook, V. Levin, and S. Rajamani, "SLAM and static driver verifier: Technology transfer of formal methods inside Microsoft," in *Integrated Formal Methods* (E. Boiten, J. Derrick, and G. Smith, eds.), vol. 2999 of *Lecture Notes in Computer Science*, pp. 1–20, Springer Berlin Heidelberg, 2004.

[114] E. Clarke and E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Logics of Programs* (D. Kozen, ed.), vol. 131 of *Lecture Notes in Computer Science*, pp. 52–71, Springer Berlin Heidelberg, 1982.

[115] E. Clarke, E. Emerson, and A. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, pp. 244–263, 1986.

[116] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Information and Computation*, vol. 98, pp. 142 – 170, 1992.

[117] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, pp. 677–691, 1986.

[118] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *Journal of the ACM*, vol. 50, pp. 752–794, 2003.

[119] T. Han and J.-P. Katoen, "Counterexamples in probabilistic model checking," in *Tools and Algorithms for the Construction and Analysis of Systems* (O. Grumberg and M. Huth, eds.), vol. 4424 of *Lecture Notes in Computer Science*, pp. 72–86, Springer Berlin Heidelberg, 2007.

[120] M. Vardi, "Automatic verification of probabilistic concurrent finite state programs," in *26th Annual Symposium on Foundations of Computer Science*, pp. 327–338, 1985.

[121] C. Courcoubetis and M. Yannakakis, "Verifying temporal properties of finite-state probabilistic programs," *Foundations of Computer Science*, vol. 861, 1988.

[122] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time Markov chains," in *Proceedings of the 10th International Conference on Concurrency Theory*, pp. 146–161, Springer, 1999.

[123] B. Philippe and R. B. Sidje, "Transient solutions of Markov processes by Krylov subspaces," in *Computations with Markov Chains* (W. Stewart, ed.), pp. 95–119, Springer US, 1995.

[124] M. Mateescu, V. Wolf, F. Didier, and T. A. Henzinger, "Fast adaptive uniformisation of the chemical master equation," *IET Systems Biology*, vol. 4, pp. 441–452, 2010.

[125] A. P. A. van Moorsel and W. H. Sanders, "Adaptive uniformization," *Stochastic Models*, vol. 10, pp. 619–647, 1994.

[126] J. Katoen, M. Kwiatkowska, G. Norman, and D. Parker, "Faster and symbolic CTMC model checking," *Process Algebra and Performance*, vol. 2165, pp. 23–38, 2001.

[127] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: a hybrid approach," *International Journal on Software Tools for Technology Transfer*, vol. 6, no. 2, 2004.

[128] H. L. Younes and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *Computer Aided Verification* (E. Brinksma and K. G. Larsen, eds.), vol. 2404 of *Lecture Notes in Computer Science*, pp. 223–235, Springer Berlin Heidelberg, 2002.

[129] H. L. Younes, M. Kwiatkowska, G. Norman, and D. Parker, "Numerical vs. statistical probabilistic model checking," *International Journal on Software Tools for Technology Transfer*, vol. 8, pp. 216–228, 2006.

[130] S. Ross, *Stochastic Processes*. Wiley, 2004.

[131] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, "A formal analysis of Bluetooth device discovery," *International Journal on Software Tools for Technology Transfer*, vol. 8, pp. 621–632, 2006.

[132] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol," *Formal Aspects of Computing*, vol. 14, pp. 295–318, 2003.

[133] M. Kwiatkowska, G. Norman, and D. Parker, "Using probabilistic model checking in systems biology," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, pp. 14–21, 2008.

[134] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn, "Probabilistic model checking of complex biological pathways," *Theoretical Computer Science*, vol. 319, pp. 239–257, 2008.

[135] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *SFM'07*, vol. 4486 of *LNCS (Tutorial Volume)*, pp. 220–270, Springer, 2007.

[136] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker," in *Second International Conference on the Quantitative Evaluation of Systems*, pp. 243–244, 2005.

[137] M. Schwarick, M. Heiner, and C. Rohr, "MARCIE - model checking and reachability analysis done efficiently," in *Eighth International Conference on Quantitative Evaluation of Systems*, pp. 91–100, 2011.

[138] M. Tribastone, A. Duguid, and S. Gilmore, "The PEPA eclipse plugin," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, pp. 28–33, 2009.

[139] P. Ballarini, B. Barbot, M. Duflot, S. Haddad, and N. Pekergin, "HASL: A new approach for performance evaluation and model checking from concepts to experimentation," *Performance Evaluation*, vol. 90, pp. 53–77, 2015.

[140] M. Lapin, L. Mikeev, and V. Wolf, "SHAVE: stochastic hybrid analysis of markov population models," in *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, pp. 311–312, ACM, 2011.

[141] H. Chandran, N. Gopalkrishnan, A. Phillips, and J. Reif, "Localized hybridization circuits," in *DNA Computing and Molecular Programming* (L. Cardelli and W. Shih, eds.), vol. 6937 of *Lecture Notes in Computer Science*, pp. 64–83, Springer Berlin Heidelberg, 2011.

[142] L. Qian and E. Winfree, "Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface," in *DNA Computing and Molecular Programming* (S. Murata and S. Kobayashi, eds.), vol. 8727 of *Lecture Notes in Computer Science*, pp. 114–131, Springer International Publishing, 2014.

[143] M. Massey, W. R. Algar, and U. J. Krull, "Fluorescence resonance energy transfer (fret) for dna biosensors: Fret pairs and förster distances for various dye–dna conjugates," *Analytica chimica acta*, vol. 568, no. 1, pp. 181–189, 2006.

[144] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.

[145] M. Mateescu, *Propagation models for biochemical reaction networks*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2011.

[146] A. P. A. van Moorsel and K. Wolter, "Numerical solution of non-homogeneous Markov processes through uniformization," in *12th European Simulation Multiconference. Simulation - Past, Present and Future*, pp. 710–717, 1998.

[147] L. Brim, M. Češka, S. Dražan, and D. Šafránek, "Exploring parameter space of stochastic biochemical systems using quantitative model checking," in *Computer Aided Verification* (N. Sharygina and H. Veith, eds.), vol. 8044 of *Lecture Notes in Computer Science*, pp. 107–123, Springer, 2013.

[148] T. Han, J.-P. Katoen, and A. Mereacre, "Approximate parameter synthesis for probabilistic time-bounded reachability," in *Proceedings of the 2008 Real-Time Systems Symposium*, pp. 173–182, IEEE Computer Society, 2008.

[149] C. Belta and L. Habets, "Controlling a class of nonlinear systems on rectangles," *IEEE Transactions on Automatic Control*, vol. 51, pp. 1749–1759, 2006.

[150] B. Sassi, M. Amin, and A. Girard, "Control of polynomial dynamical systems on rectangles," in *2013 European Control Conference*, pp. 658–663, IEEE, 2013.

[151] J.-P. Katoen, M. Z. Kwiatkowska, G. Norman, and D. Parker, "Faster and symbolic CTMC model checking," in *Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pp. 23–38, Springer, 2001.

[152] E. M. Hahn, T. Han, and L. Zhang, "Synthesis for PCTL in parametric markov decision processes," in *NASA Formal Methods* (M. Bobaru, K. Havelund, G. Holzmann, and R. Joshi, eds.), vol. 6617 of *Lecture Notes in Computer Science*, pp. 146–161, Springer, 2011.

[153] J.-P. J. Sobczak, T. G. Martin, T. Gerling, and H. Dietz, "Rapid Folding of DNA into Nanoscale Shapes at Constant Temperature," *Science*, vol. 338, pp. 1458–1461, 2012.

[154] X. Wei, J. Nangreave, S. Jiang, H. Yan, and Y. Liu, "Mapping the thermal behavior of DNA origami nanostructures," *Journal of the American Chemical Society*, vol. 135, pp. 6165–6176, 2013.

[155] Y. Ke, G. Bellot, N. V. Voigt, E. Fradkov, and W. M. Shih, "Two design strategies for enhancement of multilayer-DNA-origami folding: underwinding for specific intercalator rescue and staple-break positioning," *Chemical Science*, vol. 3, pp. 2587–2597, 2012.

[156] T. G. Martin and H. Dietz, "Magnesium-free self-assembly of multi-layer DNA objects," *Nature Communications*, vol. 3, p. 1103, 2012.

[157] J.-M. Arbona, J.-P. Aime, and J. Elezgaray, "Cooperativity in the annealing of DNA origamis," *The Journal of Chemical Physics*, vol. 138, pp. 15105–15110, 2013.

[158] C. Svaneborg, H. Fellermann, and S. Rasmussen, "DNA self-assembly and computation studied with a coarse-grained dynamic bonded model," in *DNA Computing and Molecular Programming* (D. Stefanovic and A. Turberfield, eds.), vol. 7433 of *Lecture Notes in Computer Science*, pp. 123–134, Springer, 2012.

[159] D. V. Pyshnyi and E. M. Ivanova, "Thermodynamic parameters of coaxial stacking on stacking hybridization of oligodeoxyribonucleotides," *Russian Chemical Bulletin*, vol. 51, pp. 1145–1155, 2002.

[160] R. Owczarzy, B. G. Moreira, Y. You, M. A. Behlke, and J. A. Walder, "Predicting stability of DNA duplexes in solutions containing magnesium and monovalent cations," *Biochemistry*, vol. 47, pp. 5336–5353, 2008.

[161] Lord Rayleigh, "On the problem of random vibrations, and of random flights in one, two, or three dimensions," *Philosophical Magazine*, vol. 37, pp. 321–347, 1919.

[162] S. Chandrasekhar, "Stochastic problems in physics and astronomy," *Reviews of modern physics*, vol. 15, pp. 1–89, 1943.

[163] M. E. Fisher, "Effect of excluded volume on phase transitions in biopolymers," *The Journal of Chemical Physics*, vol. 45, pp. 1469–1473, 1966.

[164] N. L. Goddard, G. Bonnet, O. Krichevsky, and A. Libchaber, "Sequence dependent rigidity of single stranded DNA," *Physical Review Letters*, vol. 85, pp. 2400–2403, 2000.

[165] W. Saenger, *Principles of Nucleic Acid Structure*. Springer, 1984.

[166] S. B. Smith, Y. Cui, and C. Bustamante, "Overstretching B-DNA: the elastic response of individual double-stranded and single-stranded DNA molecules," *Science*, vol. 271, pp. 795–9, 1996.

[167] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[168] K. E. Dunn, *DNA origami assembly*. PhD thesis, University of Oxford, 2014.

[169] T. E. Cloutier and J. Widom, "Spontaneous sharp bending of double-stranded DNA," *Molecular Cell*, vol. 14, pp. 355–362, 2004.

[170] A. P. Fields, E. A. Meyer, and A. E. Cohen, "Euler buckling and nonlinear kinking of double-stranded DNA," *Nucleic Acids Research*, vol. 41, pp. 9881–9890, 2013.

[171] B. Barbot and M. Kwiatkowska, "On quantitative modelling and verification of DNA walker circuits using stochastic petri nets," vol. 9115 of *Lecture Notes in Computer Science*, pp. 1–32, Springer International Publishing, 2015.

Appendix A

DNA walker PRISM code

control.pm

```
ctmc
const double failureRate=0.3;
module walker

stator1 : [0 .. 1] init 1;
stator2 : [0 .. 1] init 0;
stator3 : [0 .. 1] init 0;
stator4 : [0 .. 1] init 0;
stator5 : [0 .. 1] init 0;
stator6 : [0 .. 1] init 0;
stator7 : [0 .. 1] init 0;
stator8 : [0 .. 1] init 0;

w1 : [0 .. 8] init 1; // w1=0 is sinkstate for deadlocks

[step] w1=1 & stator2=0 -> 0.0029999999999999996 : (w1'=2) & (stator2'=1);
[step] w1=1 & stator3=0 -> 5.9999999999999995E-5 : (w1'=3) & (stator3'=1);
[step] w1=1 & stator4=0 -> 2.9999999999999997E-5 : (w1'=4) & (stator4'=1);

[step] w1=2 & stator1=0 -> 0.009 : (w1'=1) & (stator1'=1);
[step] w1=2 & stator3=0 -> 0.009 : (w1'=3) & (stator3'=1);
[step] w1=2 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=2 & stator5=0 -> 8.999999999999999E-5 : (w1'=5) & (stator5'=1);

[step] w1=3 & stator1=0 -> 1.7999999999999998E-4 : (w1'=1) & (stator1'=1);
[step] w1=3 & stator2=0 -> 0.009 : (w1'=2) & (stator2'=1);
[step] w1=3 & stator4=0 -> 0.009 : (w1'=4) & (stator4'=1);
[step] w1=3 & stator5=0 -> 1.7999999999999998E-4 : (w1'=5) & (stator5'=1);
[step] w1=3 & stator6=0 -> 8.999999999999999E-5 : (w1'=6) & (stator6'=1);

[step] w1=4 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=4 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=4 & stator3=0 -> 0.009 : (w1'=3) & (stator3'=1);
[step] w1=4 & stator5=0 -> 0.009 : (w1'=5) & (stator5'=1);
[step] w1=4 & stator6=0 -> 1.7999999999999998E-4 : (w1'=6) & (stator6'=1);
[step] w1=4 & stator7=0 -> 8.999999999999999E-5 : (w1'=7) & (stator7'=1);

[step] w1=5 & stator2=0 -> 8.999999999999999E-5 : (w1'=2) & (stator2'=1);
```

```
[step] w1=5 & stator3=0 -> 1.7999999999999998E-4 : (w1'=3) & (stator3'=1);
[step] w1=5 & stator4=0 -> 0.009 : (w1'=4) & (stator4'=1);
[step] w1=5 & stator6=0 -> 0.009 : (w1'=6) & (stator6'=1);
[step] w1=5 & stator7=0 -> 1.7999999999999998E-4 : (w1'=7) & (stator7'=1);
[step] w1=5 & stator8=0 -> 8.999999999999999E-6 : (w1'=8) & (stator8'=1);

[step] w1=6 & stator3=0 -> 8.999999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=6 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=6 & stator5=0 -> 0.009 : (w1'=5) & (stator5'=1);
[step] w1=6 & stator7=0 -> 0.009 : (w1'=7) & (stator7'=1);
[step] w1=6 & stator8=0 -> 1.7999999999999997E-5 : (w1'=8) & (stator8'=1);

[step] w1=7 & stator4=0 -> 8.999999999999999E-5 : (w1'=4) & (stator4'=1);
[step] w1=7 & stator5=0 -> 1.7999999999999998E-4 : (w1'=5) & (stator5'=1);
[step] w1=7 & stator6=0 -> 0.009 : (w1'=6) & (stator6'=1);
[step] w1=7 & stator8=0 -> 9.0E-4 : (w1'=8) & (stator8'=1);
[] w1 = 1 ->.000000001 :  (w1'=1);
[] w1 = 2 ->.000000001 :  (w1'=2);
[] w1 = 3 ->.000000001 :  (w1'=3);
[] w1 = 4 ->.000000001 :  (w1'=4);
[] w1 = 5 ->.000000001 :  (w1'=5);
[] w1 = 6 ->.000000001 :  (w1'=6);
[] w1 = 7 ->.000000001 :  (w1'=7);
[] w1 = 8 ->.000000001 :  (w1'=8);
endmodule

rewards "steps"
[step] true : 1;
endrewards

rewards "time"
 true : 1;
endrewards


label "deadlockUser" =
( w1=1 & stator2=1 & stator3=1 & stator4=1)
| ( w1=2 & stator1=1 & stator3=1 & stator4=1 & stator5=1)
| ( w1=3 & stator1=1 & stator2=1 & stator4=1 & stator5=1 & stator6=1)
| ( w1=4 & stator1=1 & stator2=1 & stator3=1 & stator5=1 & stator6=1 & stator7=1)
| ( w1=5 & stator2=1 & stator3=1 & stator4=1 & stator6=1 & stator7=1 & stator8=1)
| ( w1=6 & stator3=1 & stator4=1 & stator5=1 & stator7=1 & stator8=1)
| ( w1=7 & stator4=1 & stator5=1 & stator6=1 & stator8=1);
```

control.pctl

```
P = ? [F[12000,12000] (w1=2) ]
P = ? [F[12000,12000] (w1=8) ]
P = ? [F[12000,12000] "deadlockUser" ]
R{"steps"}=? [ C<=200*60 ]
```

track12Block2.pm

```
ctmc
```

```
const double failureRate=0.3;

  module walker

stator1 : [0 .. 1] init 1;
stator2 : [0 .. 1] init 0;
stator3 : [0 .. 1] init 0;
stator4 : [0 .. 1] init 0;
stator5 : [0 .. 1] init 1;
stator6 : [0 .. 1] init 1;
stator7 : [0 .. 1] init 0;
stator8 : [0 .. 1] init 0;
stator9 : [0 .. 1] init 0;
stator10 : [0 .. 1] init 0;
stator11 : [0 .. 1] init 0;
stator12 : [0 .. 1] init 0;

w1 : [0 .. 12] init 1; // w1=0 is sinkstate for deadlocks


blockade5 : [0 .. 1] init 0;
blockade6 : [0 .. 1] init 0;

[block5]  blockade5=0  ->1000000.0 * failureRate  : (blockade5'=1) & (stator5'=0)
+  1000000.0 * (1.0 - failureRate ) : (blockade5'=1);
[block6]  blockade6=0  ->1000000.0 * failureRate  : (blockade6'=1) & (stator6'=0)
+  1000000.0 * (1.0 - failureRate ) : (blockade6'=1);


[step] w1=1 & stator2=0 -> 0.0029999999999999996 : (w1'=2) & (stator2'=1);
[step] w1=1 & stator3=0 -> 5.9999999999999995E-5 : (w1'=3) & (stator3'=1);
[step] w1=1 & stator4=0 -> 2.9999999999999997E-5 : (w1'=4) & (stator4'=1);
[step] w1=1 & stator5=0 -> 2.9999999999999997E-5 : (w1'=5) & (stator5'=1);
[step] w1=1 & stator9=0 -> 2.9999999999999997E-5 : (w1'=9) & (stator9'=1);

[step] w1=2 & stator1=0 -> 0.009 : (w1'=1) & (stator1'=1);
[step] w1=2 & stator3=0 -> 0.009 : (w1'=3) & (stator3'=1);
[step] w1=2 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=2 & stator5=0 -> 1.7999999999999998E-4 : (w1'=5) & (stator5'=1);
[step] w1=2 & stator6=0 -> 8.999999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=2 & stator9=0 -> 1.7999999999999998E-4 : (w1'=9) & (stator9'=1);
[step] w1=2 & stator10=0 -> 8.999999999999999E-5 : (w1'=10) & (stator10'=1);

[step] w1=3 & stator1=0 -> 1.7999999999999998E-4 : (w1'=1) & (stator1'=1);
[step] w1=3 & stator2=0 -> 0.009 : (w1'=2) & (stator2'=1);
[step] w1=3 & stator4=0 -> 0.009 : (w1'=4) & (stator4'=1);
[step] w1=3 & stator5=0 -> 1.7999999999999998E-4 : (w1'=5) & (stator5'=1);
[step] w1=3 & stator6=0 -> 8.999999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=3 & stator7=0 -> 8.999999999999999E-5 : (w1'=7) & (stator7'=1);
[step] w1=3 & stator9=0 -> 1.7999999999999998E-4 : (w1'=9) & (stator9'=1);
[step] w1=3 & stator10=0 -> 8.999999999999999E-5 : (w1'=10) & (stator10'=1);
[step] w1=3 & stator11=0 -> 8.999999999999999E-5 : (w1'=11) & (stator11'=1);

[step] w1=4 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=4 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=4 & stator3=0 -> 0.009 : (w1'=3) & (stator3'=1);
[step] w1=4 & stator5=0 -> 0.009 : (w1'=5) & (stator5'=1);
```

```
[step] w1=4 & stator6=0 -> 1.7999999999999998E-4 : (w1'=6) & (stator6'=1);
[step] w1=4 & stator7=0 -> 8.999999999999999E-5 : (w1'=7) & (stator7'=1);
[step] w1=4 & stator9=0 -> 0.009 : (w1'=9) & (stator9'=1);
[step] w1=4 & stator10=0 -> 1.7999999999999998E-4 : (w1'=10) & (stator10'=1);
[step] w1=4 & stator11=0 -> 8.999999999999999E-5 : (w1'=11) & (stator11'=1);

[step] w1=5 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=5 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=5 & stator3=0 -> 1.7999999999999998E-4 : (w1'=3) & (stator3'=1);
[step] w1=5 & stator4=0 -> 0.009 : (w1'=4) & (stator4'=1);
[step] w1=5 & stator6=0 -> 0.009 : (w1'=6) & (stator6'=1);
[step] w1=5 & stator7=0 -> 1.7999999999999998E-4 : (w1'=7) & (stator7'=1);
[step] w1=5 & stator8=0 -> 8.999999999999999E-6 : (w1'=8) & (stator8'=1);
[step] w1=5 & stator9=0 -> 1.7999999999999998E-4 : (w1'=9) & (stator9'=1);
[step] w1=5 & stator10=0 -> 8.999999999999999E-5 : (w1'=10) & (stator10'=1);
[step] w1=5 & stator11=0 -> 8.999999999999999E-5 : (w1'=11) & (stator11'=1);

[step] w1=6 & stator2=0 -> 8.999999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=6 & stator3=0 -> 8.999999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=6 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=6 & stator5=0 -> 0.009 : (w1'=5) & (stator5'=1);
[step] w1=6 & stator7=0 -> 0.009 : (w1'=7) & (stator7'=1);
[step] w1=6 & stator8=0 -> 1.7999999999999997E-5 : (w1'=8) & (stator8'=1);
[step] w1=6 & stator9=0 -> 8.999999999999999E-5 : (w1'=9) & (stator9'=1);
[step] w1=6 & stator10=0 -> 8.999999999999999E-5 : (w1'=10) & (stator10'=1);

[step] w1=7 & stator3=0 -> 8.999999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=7 & stator4=0 -> 8.999999999999999E-5 : (w1'=4) & (stator4'=1);
[step] w1=7 & stator5=0 -> 1.7999999999999998E-4 : (w1'=5) & (stator5'=1);
[step] w1=7 & stator6=0 -> 0.009 : (w1'=6) & (stator6'=1);
[step] w1=7 & stator8=0 -> 9.0E-4 : (w1'=8) & (stator8'=1);
[step] w1=7 & stator9=0 -> 8.999999999999999E-5 : (w1'=9) & (stator9'=1);

[step] w1=9 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=9 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=9 & stator3=0 -> 1.7999999999999998E-4 : (w1'=3) & (stator3'=1);
[step] w1=9 & stator4=0 -> 0.009 : (w1'=4) & (stator4'=1);
[step] w1=9 & stator5=0 -> 1.7999999999999998E-4 : (w1'=5) & (stator5'=1);
[step] w1=9 & stator6=0 -> 8.999999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=9 & stator7=0 -> 8.999999999999999E-5 : (w1'=7) & (stator7'=1);
[step] w1=9 & stator10=0 -> 0.009 : (w1'=10) & (stator10'=1);
[step] w1=9 & stator11=0 -> 1.7999999999999998E-4 : (w1'=11) & (stator11'=1);
[step] w1=9 & stator12=0 -> 8.999999999999999E-6 : (w1'=12) & (stator12'=1);

[step] w1=10 & stator2=0 -> 8.999999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=10 & stator3=0 -> 8.999999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=10 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=10 & stator5=0 -> 8.999999999999999E-5 : (w1'=5) & (stator5'=1);
[step] w1=10 & stator6=0 -> 8.999999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=10 & stator9=0 -> 0.009 : (w1'=9) & (stator9'=1);
[step] w1=10 & stator11=0 -> 0.009 : (w1'=11) & (stator11'=1);
[step] w1=10 & stator12=0 -> 1.7999999999999997E-5 : (w1'=12) & (stator12'=1);

[step] w1=11 & stator3=0 -> 8.999999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=11 & stator4=0 -> 8.999999999999999E-5 : (w1'=4) & (stator4'=1);
[step] w1=11 & stator5=0 -> 8.999999999999999E-5 : (w1'=5) & (stator5'=1);
[step] w1=11 & stator9=0 -> 1.7999999999999998E-4 : (w1'=9) & (stator9'=1);
```

```
[step] w1=11 & stator10=0 -> 0.009 : (w1'=10) & (stator10'=1);
[step] w1=11 & stator12=0 -> 9.0E-4 : (w1'=12) & (stator12'=1);
[] w1 = 1 ->.000000001 :  (w1'=1);
[] w1 = 2 ->.000000001 :  (w1'=2);
[] w1 = 3 ->.000000001 :  (w1'=3);
[] w1 = 4 ->.000000001 :  (w1'=4);
[] w1 = 5 ->.000000001 :  (w1'=5);
[] w1 = 6 ->.000000001 :  (w1'=6);
[] w1 = 7 ->.000000001 :  (w1'=7);
[] w1 = 8 ->.000000001 :  (w1'=8);
[] w1 = 9 ->.000000001 :  (w1'=9);
[] w1 = 10 ->.000000001 :  (w1'=10);
[] w1 = 11 ->.000000001 :  (w1'=11);
[] w1 = 12 ->.000000001 :  (w1'=12);
endmodule

rewards "steps"
[step] true : 1;
endrewards

rewards "time"
 true : 1;
endrewards

rewards "blocked" // time spend in blocked anchorages
w1 =5 | w1 =6 : 1;
endrewards

label "deadlockUser" =
( w1=1 & stator2=1 & stator3=1 & stator4=1 & stator5=1 & stator9=1)
| ( w1=2 & stator1=1 & stator3=1 & stator4=1 & stator5=1 & stator6=1 & stator9=1 & stator10=1)
| ( w1=3 & stator1=1 & stator2=1 & stator4=1 & stator5=1 & stator6=1 & stator7=1 & stator9=1
& stator10=1 & stator11=1)
| ( w1=4 & stator1=1 & stator2=1 & stator3=1 & stator5=1 & stator6=1 & stator7=1 & stator9=1
& stator10=1 & stator11=1)
| ( w1=5 & stator1=1 & stator2=1 & stator3=1 & stator4=1 & stator6=1 & stator7=1 & stator8=1
& stator9=1 & stator10=1 & stator11=1)
| ( w1=6 & stator2=1 & stator3=1 & stator4=1 & stator5=1 & stator7=1 & stator8=1 & stator9=1
& stator10=1)
| ( w1=7 & stator3=1 & stator4=1 & stator5=1 & stator6=1 & stator8=1 & stator9=1)
| ( w1=9 & stator1=1 & stator2=1 & stator3=1 & stator4=1 & stator5=1 & stator6=1 & stator7=1
& stator10=1 & stator11=1 & stator12=1)
| ( w1=10 & stator2=1 & stator3=1 & stator4=1 & stator5=1 & stator6=1 & stator9=1 & stator11=1
& stator12=1)
| ( w1=11 & stator3=1 & stator4=1 & stator5=1 & stator9=1 & stator10=1 & stator12=1)
;
```

track12Block2.pctl

```
P = ? [F[12000,12000] (w1=8)|(w1=12) ]
P = ? [F[12000,12000] (w1=8) ]
P = ? [F[12000,12000] (w1=12) ]
P = ? [F[12000,12000] "deadlockUser" ]
R{"steps"}=? [ C<=200*60 ]
```

ringLL.pm

```
ctmc
  const double failureRate;
  module walker

stator1 : [0 .. 1] init 1;
stator2 : [0 .. 1] init 1;
stator3 : [0 .. 1] init 1;
stator4 : [0 .. 1] init 1;
stator5 : [0 .. 1] init 1;
stator6 : [0 .. 1] init 0;
stator7 : [0 .. 1] init 0;
stator8 : [0 .. 1] init 0;
stator9 : [0 .. 1] init 0;
stator10 : [0 .. 1] init 0;
stator11 : [0 .. 1] init 0;
stator12 : [0 .. 1] init 0;
stator13 : [0 .. 1] init 0;
stator14 : [0 .. 1] init 1;
stator15 : [0 .. 1] init 1;
stator16 : [0 .. 1] init 0;
stator17 : [0 .. 1] init 0;
stator18 : [0 .. 1] init 0;
stator19 : [0 .. 1] init 0;
stator20 : [0 .. 1] init 0;
stator21 : [0 .. 1] init 0;

w1 : [0 .. 21] init 1; // w1=0 is sinkstate for deadlocks

blockade2 : [0 .. 1] init 0;
blockade3 : [0 .. 1] init 0;
blockade4 : [0 .. 1] init 0;
blockade5 : [0 .. 1] init 0;
blockade14 : [0 .. 1] init 0;
blockade15 : [0 .. 1] init 0;

[block2]  blockade2=0  ->1000000.0 * failureRate  : (blockade2'=1) & (stator2'=0)
+  1000000.0 * (1.0 - failureRate ) : (blockade2'=1);
[block3]  blockade3=0  ->1000000.0 * failureRate  : (blockade3'=1) & (stator3'=0)
+  1000000.0 * (1.0 - failureRate ) : (blockade3'=1);
[block4]  blockade4=0  ->1000000.0 * failureRate  : (blockade4'=1) & (stator4'=0)
+  1000000.0 * (1.0 - failureRate ) : (blockade4'=1);
[block5]  blockade5=0  ->1000000.0 * failureRate  : (blockade5'=1) & (stator5'=0)
+  1000000.0 * (1.0 - failureRate ) : (blockade5'=1);
[block14]  blockade14=0  ->1000000.0 * failureRate  : (blockade14'=1) & (stator14'=0)
+  1000000.0 * (1.0 - failureRate ) : (blockade14'=1);
[block15]  blockade15=0  ->1000000.0 * failureRate  : (blockade15'=1) & (stator15'=0)
+  1000000.0 * (1.0 - failureRate ) : (blockade15'=1);

[step] w1=1 & stator2=0 -> 0.0029999999999999996 : (w1'=2) & (stator2'=1);
[step] w1=1 & stator3=0 -> 5.9999999999999995E-5 : (w1'=3) & (stator3'=1);
[step] w1=1 & stator4=0 -> 2.9999999999999997E-5 : (w1'=4) & (stator4'=1);
[step] w1=1 & stator5=0 -> 5.9999999999999995E-5 : (w1'=5) & (stator5'=1);
[step] w1=1 & stator6=0 -> 2.9999999999999997E-5 : (w1'=6) & (stator6'=1);
[step] w1=1 & stator7=0 -> 5.999999999999999E-6 : (w1'=7) & (stator7'=1);
[step] w1=1 & stator8=0 -> 2.9999999999999997E-5 : (w1'=8) & (stator8'=1);
```

```
[step] w1=1 & stator9=0 -> 2.9999999999999997E-5 : (w1'=9) & (stator9'=1);
[step] w1=1 & stator10=0 -> 2.9999999999999997E-5 : (w1'=10) & (stator10'=1);
[step] w1=1 & stator11=0 -> 2.9999999999999997E-5 : (w1'=11) & (stator11'=1);
[step] w1=1 & stator12=0 -> 5.9999999999999995E-5 : (w1'=12) & (stator12'=1);
[step] w1=1 & stator13=0 -> 0.00299999999999999996 : (w1'=13) & (stator13'=1);
[step] w1=1 & stator14=0 -> 2.9999999999999997E-5 : (w1'=14) & (stator14'=1);
[step] w1=1 & stator15=0 -> 5.9999999999999995E-5 : (w1'=15) & (stator15'=1);
[step] w1=1 & stator16=0 -> 2.9999999999999997E-5 : (w1'=16) & (stator16'=1);
[step] w1=1 & stator17=0 -> 5.999999999999999E-6 : (w1'=17) & (stator17'=1);
[step] w1=1 & stator18=0 -> 2.9999999999999997E-5 : (w1'=18) & (stator18'=1);
[step] w1=1 & stator19=0 -> 2.9999999999999997E-5 : (w1'=19) & (stator19'=1);
[step] w1=1 & stator20=0 -> 2.9999999999999997E-5 : (w1'=20) & (stator20'=1);
[step] w1=1 & stator21=0 -> 2.9999999999999997E-5 : (w1'=21) & (stator21'=1);

[step] w1=2 & stator1=0 -> 0.009 : (w1'=1) & (stator1'=1);
[step] w1=2 & stator3=0 -> 0.009 : (w1'=3) & (stator3'=1);
[step] w1=2 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=2 & stator5=0 -> 1.7999999999999998E-4 : (w1'=5) & (stator5'=1);
[step] w1=2 & stator6=0 -> 1.7999999999999998E-4 : (w1'=6) & (stator6'=1);
[step] w1=2 & stator7=0 -> 8.999999999999999E-6 : (w1'=7) & (stator7'=1);
[step] w1=2 & stator8=0 -> 8.999999999999999E-5 : (w1'=8) & (stator8'=1);
[step] w1=2 & stator9=0 -> 8.999999999999999E-5 : (w1'=9) & (stator9'=1);
[step] w1=2 & stator10=0 -> 8.999999999999999E-5 : (w1'=10) & (stator10'=1);
[step] w1=2 & stator11=0 -> 8.999999999999999E-5 : (w1'=11) & (stator11'=1);
[step] w1=2 & stator12=0 -> 8.999999999999999E-5 : (w1'=12) & (stator12'=1);
[step] w1=2 & stator13=0 -> 1.7999999999999998E-4 : (w1'=13) & (stator13'=1);
[step] w1=2 & stator14=0 -> 8.999999999999999E-5 : (w1'=14) & (stator14'=1);
[step] w1=2 & stator15=0 -> 8.999999999999999E-5 : (w1'=15) & (stator15'=1);
[step] w1=2 & stator16=0 -> 8.999999999999999E-5 : (w1'=16) & (stator16'=1);
[step] w1=2 & stator17=0 -> 8.999999999999999E-6 : (w1'=17) & (stator17'=1);
[step] w1=2 & stator18=0 -> 1.7999999999999998E-4 : (w1'=18) & (stator18'=1);
[step] w1=2 & stator19=0 -> 8.999999999999999E-5 : (w1'=19) & (stator19'=1);
[step] w1=2 & stator20=0 -> 1.7999999999999998E-4 : (w1'=20) & (stator20'=1);
[step] w1=2 & stator21=0 -> 1.7999999999999998E-4 : (w1'=21) & (stator21'=1);

[step] w1=3 & stator1=0 -> 1.7999999999999998E-4 : (w1'=1) & (stator1'=1);
[step] w1=3 & stator2=0 -> 0.009 : (w1'=2) & (stator2'=1);
[step] w1=3 & stator4=0 -> 0.009 : (w1'=4) & (stator4'=1);
[step] w1=3 & stator5=0 -> 1.7999999999999998E-4 : (w1'=5) & (stator5'=1);
[step] w1=3 & stator6=0 -> 8.999999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=3 & stator7=0 -> 8.999999999999999E-6 : (w1'=7) & (stator7'=1);
[step] w1=3 & stator8=0 -> 8.999999999999999E-5 : (w1'=8) & (stator8'=1);
[step] w1=3 & stator13=0 -> 8.999999999999999E-5 : (w1'=13) & (stator13'=1);
[step] w1=3 & stator17=0 -> 8.999999999999999E-6 : (w1'=17) & (stator17'=1);
[step] w1=3 & stator18=0 -> 8.999999999999999E-5 : (w1'=18) & (stator18'=1);
[step] w1=3 & stator19=0 -> 1.7999999999999998E-4 : (w1'=19) & (stator19'=1);
[step] w1=3 & stator20=0 -> 1.7999999999999998E-4 : (w1'=20) & (stator20'=1);
[step] w1=3 & stator21=0 -> 0.009 : (w1'=21) & (stator21'=1);

[step] w1=4 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=4 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=4 & stator3=0 -> 0.009 : (w1'=3) & (stator3'=1);
[step] w1=4 & stator5=0 -> 0.009 : (w1'=5) & (stator5'=1);
[step] w1=4 & stator6=0 -> 1.7999999999999998E-4 : (w1'=6) & (stator6'=1);
[step] w1=4 & stator7=0 -> 8.999999999999999E-6 : (w1'=7) & (stator7'=1);
[step] w1=4 & stator8=0 -> 8.999999999999999E-5 : (w1'=8) & (stator8'=1);
[step] w1=4 & stator13=0 -> 8.999999999999999E-5 : (w1'=13) & (stator13'=1);
```

```
[step] w1=4 & stator18=0 -> 8.9999999999999E-5 : (w1'=18) & (stator18'=1);
[step] w1=4 & stator19=0 -> 8.9999999999999E-5 : (w1'=19) & (stator19'=1);
[step] w1=4 & stator20=0 -> 1.7999999999999998E-4 : (w1'=20) & (stator20'=1);
[step] w1=4 & stator21=0 -> 1.7999999999999998E-4 : (w1'=21) & (stator21'=1);

[step] w1=5 & stator1=0 -> 1.7999999999999998E-4 : (w1'=1) & (stator1'=1);
[step] w1=5 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=5 & stator3=0 -> 1.7999999999999998E-4 : (w1'=3) & (stator3'=1);
[step] w1=5 & stator4=0 -> 0.009 : (w1'=4) & (stator4'=1);
[step] w1=5 & stator6=0 -> 0.009 : (w1'=6) & (stator6'=1);
[step] w1=5 & stator7=0 -> 1.7999999999999997E-5 : (w1'=7) & (stator7'=1);
[step] w1=5 & stator8=0 -> 8.9999999999999E-5 : (w1'=8) & (stator8'=1);
[step] w1=5 & stator9=0 -> 8.9999999999999E-5 : (w1'=9) & (stator9'=1);
[step] w1=5 & stator13=0 -> 8.9999999999999E-5 : (w1'=13) & (stator13'=1);
[step] w1=5 & stator20=0 -> 8.9999999999999E-5 : (w1'=20) & (stator20'=1);
[step] w1=5 & stator21=0 -> 8.9999999999999E-5 : (w1'=21) & (stator21'=1);

[step] w1=6 & stator1=0 -> 8.9999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=6 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=6 & stator3=0 -> 8.9999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=6 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=6 & stator5=0 -> 0.009 : (w1'=5) & (stator5'=1);
[step] w1=6 & stator7=0 -> 9.0E-4 : (w1'=7) & (stator7'=1);
[step] w1=6 & stator8=0 -> 1.7999999999999998E-4 : (w1'=8) & (stator8'=1);
[step] w1=6 & stator9=0 -> 8.9999999999999E-5 : (w1'=9) & (stator9'=1);
[step] w1=6 & stator10=0 -> 8.9999999999999E-5 : (w1'=10) & (stator10'=1);
[step] w1=6 & stator13=0 -> 8.9999999999999E-5 : (w1'=13) & (stator13'=1);
[step] w1=6 & stator21=0 -> 8.9999999999999E-5 : (w1'=21) & (stator21'=1);

[step] w1=8 & stator1=0 -> 8.9999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=8 & stator2=0 -> 8.9999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=8 & stator3=0 -> 8.9999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=8 & stator4=0 -> 8.9999999999999E-5 : (w1'=4) & (stator4'=1);
[step] w1=8 & stator5=0 -> 8.9999999999999E-5 : (w1'=5) & (stator5'=1);
[step] w1=8 & stator6=0 -> 1.7999999999999998E-4 : (w1'=6) & (stator6'=1);
[step] w1=8 & stator7=0 -> 9.0E-4 : (w1'=7) & (stator7'=1);
[step] w1=8 & stator9=0 -> 0.009 : (w1'=9) & (stator9'=1);
[step] w1=8 & stator10=0 -> 1.7999999999999998E-4 : (w1'=10) & (stator10'=1);
[step] w1=8 & stator11=0 -> 1.7999999999999998E-4 : (w1'=11) & (stator11'=1);
[step] w1=8 & stator12=0 -> 8.9999999999999E-5 : (w1'=12) & (stator12'=1);
[step] w1=8 & stator13=0 -> 1.7999999999999998E-4 : (w1'=13) & (stator13'=1);
[step] w1=8 & stator14=0 -> 8.9999999999999E-5 : (w1'=14) & (stator14'=1);

[step] w1=9 & stator1=0 -> 8.9999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=9 & stator2=0 -> 8.9999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=9 & stator5=0 -> 8.9999999999999E-5 : (w1'=5) & (stator5'=1);
[step] w1=9 & stator6=0 -> 8.9999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=9 & stator7=0 -> 1.7999999999999997E-5 : (w1'=7) & (stator7'=1);
[step] w1=9 & stator8=0 -> 0.009 : (w1'=8) & (stator8'=1);
[step] w1=9 & stator10=0 -> 0.009 : (w1'=10) & (stator10'=1);
[step] w1=9 & stator11=0 -> 1.7999999999999998E-4 : (w1'=11) & (stator11'=1);
[step] w1=9 & stator12=0 -> 1.7999999999999998E-4 : (w1'=12) & (stator12'=1);
[step] w1=9 & stator13=0 -> 8.9999999999999E-5 : (w1'=13) & (stator13'=1);
[step] w1=9 & stator14=0 -> 8.9999999999999E-5 : (w1'=14) & (stator14'=1);

[step] w1=10 & stator1=0 -> 8.9999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=10 & stator2=0 -> 8.9999999999999E-5 : (w1'=2) & (stator2'=1);
```

```
[step] w1=10 & stator6=0 -> 8.9999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=10 & stator7=0 -> 8.9999999999999E-6 : (w1'=7) & (stator7'=1);
[step] w1=10 & stator8=0 -> 1.7999999999999998E-4 : (w1'=8) & (stator8'=1);
[step] w1=10 & stator9=0 -> 0.009 : (w1'=9) & (stator9'=1);
[step] w1=10 & stator11=0 -> 0.009 : (w1'=11) & (stator11'=1);
[step] w1=10 & stator12=0 -> 1.7999999999999998E-4 : (w1'=12) & (stator12'=1);
[step] w1=10 & stator13=0 -> 1.7999999999999998E-4 : (w1'=13) & (stator13'=1);
[step] w1=10 & stator14=0 -> 1.7999999999999998E-4 : (w1'=14) & (stator14'=1);
[step] w1=10 & stator15=0 -> 8.9999999999999E-5 : (w1'=15) & (stator15'=1);
[step] w1=11 & stator1=0 -> 8.9999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=11 & stator2=0 -> 8.9999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=11 & stator7=0 -> 8.9999999999999E-6 : (w1'=7) & (stator7'=1);
[step] w1=11 & stator8=0 -> 1.7999999999999998E-4 : (w1'=8) & (stator8'=1);
[step] w1=11 & stator9=0 -> 1.7999999999999998E-4 : (w1'=9) & (stator9'=1);
[step] w1=11 & stator10=0 -> 0.009 : (w1'=10) & (stator10'=1);
[step] w1=11 & stator12=0 -> 0.009 : (w1'=12) & (stator12'=1);
[step] w1=11 & stator13=0 -> 1.7999999999999998E-4 : (w1'=13) & (stator13'=1);
[step] w1=11 & stator14=0 -> 1.7999999999999998E-4 : (w1'=14) & (stator14'=1);
[step] w1=11 & stator15=0 -> 8.9999999999999E-5 : (w1'=15) & (stator15'=1);
[step] w1=11 & stator16=0 -> 8.9999999999999E-5 : (w1'=16) & (stator16'=1);

[step] w1=12 & stator1=0 -> 1.7999999999999998E-4 : (w1'=1) & (stator1'=1);
[step] w1=12 & stator2=0 -> 8.9999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=12 & stator7=0 -> 8.9999999999999E-6 : (w1'=7) & (stator7'=1);
[step] w1=12 & stator8=0 -> 8.9999999999999E-5 : (w1'=8) & (stator8'=1);
[step] w1=12 & stator9=0 -> 1.7999999999999998E-4 : (w1'=9) & (stator9'=1);
[step] w1=12 & stator10=0 -> 1.7999999999999998E-4 : (w1'=10) & (stator10'=1);
[step] w1=12 & stator11=0 -> 0.009 : (w1'=11) & (stator11'=1);
[step] w1=12 & stator13=0 -> 0.009 : (w1'=13) & (stator13'=1);
[step] w1=12 & stator14=0 -> 0.009 : (w1'=14) & (stator14'=1);
[step] w1=12 & stator15=0 -> 1.7999999999999998E-4 : (w1'=15) & (stator15'=1);
[step] w1=12 & stator16=0 -> 8.9999999999999E-5 : (w1'=16) & (stator16'=1);
[step] w1=12 & stator17=0 -> 8.9999999999999E-6 : (w1'=17) & (stator17'=1);
[step] w1=12 & stator18=0 -> 8.9999999999999E-5 : (w1'=18) & (stator18'=1);

[step] w1=13 & stator1=0 -> 0.009 : (w1'=1) & (stator1'=1);
[step] w1=13 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=13 & stator3=0 -> 8.9999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=13 & stator4=0 -> 8.9999999999999E-5 : (w1'=4) & (stator4'=1);
[step] w1=13 & stator5=0 -> 8.9999999999999E-5 : (w1'=5) & (stator5'=1);
[step] w1=13 & stator6=0 -> 8.9999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=13 & stator7=0 -> 8.9999999999999E-6 : (w1'=7) & (stator7'=1);
[step] w1=13 & stator8=0 -> 1.7999999999999998E-4 : (w1'=8) & (stator8'=1);
[step] w1=13 & stator9=0 -> 8.9999999999999E-5 : (w1'=9) & (stator9'=1);
[step] w1=13 & stator10=0 -> 1.7999999999999998E-4 : (w1'=10) & (stator10'=1);
[step] w1=13 & stator11=0 -> 1.7999999999999998E-4 : (w1'=11) & (stator11'=1);
[step] w1=13 & stator12=0 -> 0.009 : (w1'=12) & (stator12'=1);
[step] w1=13 & stator14=0 -> 1.7999999999999998E-4 : (w1'=14) & (stator14'=1);
[step] w1=13 & stator15=0 -> 1.7999999999999998E-4 : (w1'=15) & (stator15'=1);
[step] w1=13 & stator16=0 -> 1.7999999999999998E-4 : (w1'=16) & (stator16'=1);
[step] w1=13 & stator17=0 -> 8.9999999999999E-6 : (w1'=17) & (stator17'=1);
[step] w1=13 & stator18=0 -> 8.9999999999999E-5 : (w1'=18) & (stator18'=1);
[step] w1=13 & stator19=0 -> 8.9999999999999E-5 : (w1'=19) & (stator19'=1);
[step] w1=13 & stator20=0 -> 8.9999999999999E-5 : (w1'=20) & (stator20'=1);
[step] w1=13 & stator21=0 -> 8.9999999999999E-5 : (w1'=21) & (stator21'=1);

[step] w1=14 & stator1=0 -> 8.9999999999999E-5 : (w1'=1) & (stator1'=1);
```

```
[step] w1=14 & stator2=0 -> 8.999999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=14 & stator8=0 -> 8.999999999999999E-5 : (w1'=8) & (stator8'=1);
[step] w1=14 & stator9=0 -> 8.999999999999999E-5 : (w1'=9) & (stator9'=1);
[step] w1=14 & stator10=0 -> 1.7999999999999998E-4 : (w1'=10) & (stator10'=1);
[step] w1=14 & stator11=0 -> 1.7999999999999998E-4 : (w1'=11) & (stator11'=1);
[step] w1=14 & stator12=0 -> 0.009 : (w1'=12) & (stator12'=1);
[step] w1=14 & stator13=0 -> 1.7999999999999998E-4 : (w1'=13) & (stator13'=1);
[step] w1=14 & stator15=0 -> 0.009 : (w1'=15) & (stator15'=1);
[step] w1=14 & stator16=0 -> 1.7999999999999998E-4 : (w1'=16) & (stator16'=1);
[step] w1=14 & stator17=0 -> 8.999999999999999E-6 : (w1'=17) & (stator17'=1);
[step] w1=14 & stator18=0 -> 8.999999999999999E-5 : (w1'=18) & (stator18'=1);

[step] w1=15 & stator1=0 -> 1.7999999999999998E-4 : (w1'=1) & (stator1'=1);
[step] w1=15 & stator2=0 -> 8.999999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=15 & stator10=0 -> 8.999999999999999E-5 : (w1'=10) & (stator10'=1);
[step] w1=15 & stator11=0 -> 8.999999999999999E-5 : (w1'=11) & (stator11'=1);
[step] w1=15 & stator12=0 -> 1.7999999999999998E-4 : (w1'=12) & (stator12'=1);
[step] w1=15 & stator13=0 -> 1.7999999999999998E-4 : (w1'=13) & (stator13'=1);
[step] w1=15 & stator14=0 -> 0.009 : (w1'=14) & (stator14'=1);
[step] w1=15 & stator16=0 -> 0.009 : (w1'=16) & (stator16'=1);
[step] w1=15 & stator17=0 -> 1.7999999999999997E-5 : (w1'=17) & (stator17'=1);
[step] w1=15 & stator18=0 -> 8.999999999999999E-5 : (w1'=18) & (stator18'=1);
[step] w1=15 & stator19=0 -> 8.999999999999999E-5 : (w1'=19) & (stator19'=1);


[step] w1=16 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=16 & stator2=0 -> 8.999999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=16 & stator11=0 -> 8.999999999999999E-5 : (w1'=11) & (stator11'=1);
[step] w1=16 & stator12=0 -> 8.999999999999999E-5 : (w1'=12) & (stator12'=1);
[step] w1=16 & stator13=0 -> 1.7999999999999998E-4 : (w1'=13) & (stator13'=1);
[step] w1=16 & stator14=0 -> 1.7999999999999998E-4 : (w1'=14) & (stator14'=1);
[step] w1=16 & stator15=0 -> 0.009 : (w1'=15) & (stator15'=1);
[step] w1=16 & stator17=0 -> 9.0E-4 : (w1'=17) & (stator17'=1);
[step] w1=16 & stator18=0 -> 1.7999999999999998E-4 : (w1'=18) & (stator18'=1);
[step] w1=16 & stator19=0 -> 8.999999999999999E-5 : (w1'=19) & (stator19'=1);
[step] w1=16 & stator20=0 -> 8.999999999999999E-5 : (w1'=20) & (stator20'=1);

[step] w1=18 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=18 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=18 & stator3=0 -> 8.999999999999999E-5 : (w1'=3) & (stator3'=1);
[step] w1=18 & stator4=0 -> 8.999999999999999E-5 : (w1'=4) & (stator4'=1);
[step] w1=18 & stator12=0 -> 8.999999999999999E-5 : (w1'=12) & (stator12'=1);
[step] w1=18 & stator13=0 -> 8.999999999999999E-5 : (w1'=13) & (stator13'=1);
[step] w1=18 & stator14=0 -> 8.999999999999999E-5 : (w1'=14) & (stator14'=1);
[step] w1=18 & stator15=0 -> 8.999999999999999E-5 : (w1'=15) & (stator15'=1);
[step] w1=18 & stator16=0 -> 1.7999999999999998E-4 : (w1'=16) & (stator16'=1);
[step] w1=18 & stator17=0 -> 9.0E-4 : (w1'=17) & (stator17'=1);
[step] w1=18 & stator19=0 -> 0.009 : (w1'=19) & (stator19'=1);
[step] w1=18 & stator20=0 -> 1.7999999999999998E-4 : (w1'=20) & (stator20'=1);
[step] w1=18 & stator21=0 -> 1.7999999999999998E-4 : (w1'=21) & (stator21'=1);

[step] w1=19 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=19 & stator2=0 -> 8.999999999999999E-5 : (w1'=2) & (stator2'=1);
[step] w1=19 & stator3=0 -> 1.7999999999999998E-4 : (w1'=3) & (stator3'=1);
[step] w1=19 & stator4=0 -> 8.999999999999999E-5 : (w1'=4) & (stator4'=1);
[step] w1=19 & stator13=0 -> 8.999999999999999E-5 : (w1'=13) & (stator13'=1);
[step] w1=19 & stator15=0 -> 8.999999999999999E-5 : (w1'=15) & (stator15'=1);
```

```
[step] w1=19 & stator16=0 -> 8.999999999999999E-5 : (w1'=16) & (stator16'=1);
[step] w1=19 & stator17=0 -> 1.7999999999999997E-5 : (w1'=17) & (stator17'=1);
[step] w1=19 & stator18=0 -> 0.009 : (w1'=18) & (stator18'=1);
[step] w1=19 & stator20=0 -> 0.009 : (w1'=20) & (stator20'=1);
[step] w1=19 & stator21=0 -> 1.7999999999999998E-4 : (w1'=21) & (stator21'=1);

[step] w1=20 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=20 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=20 & stator3=0 -> 1.7999999999999998E-4 : (w1'=3) & (stator3'=1);
[step] w1=20 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=20 & stator5=0 -> 8.999999999999999E-5 : (w1'=5) & (stator5'=1);
[step] w1=20 & stator13=0 -> 8.999999999999999E-5 : (w1'=13) & (stator13'=1);
[step] w1=20 & stator16=0 -> 8.999999999999999E-5 : (w1'=16) & (stator16'=1);
[step] w1=20 & stator17=0 -> 8.999999999999999E-6 : (w1'=17) & (stator17'=1);
[step] w1=20 & stator18=0 -> 1.7999999999999998E-4 : (w1'=18) & (stator18'=1);
[step] w1=20 & stator19=0 -> 0.009 : (w1'=19) & (stator19'=1);
[step] w1=20 & stator21=0 -> 0.009 : (w1'=21) & (stator21'=1);

[step] w1=21 & stator1=0 -> 8.999999999999999E-5 : (w1'=1) & (stator1'=1);
[step] w1=21 & stator2=0 -> 1.7999999999999998E-4 : (w1'=2) & (stator2'=1);
[step] w1=21 & stator3=0 -> 0.009 : (w1'=3) & (stator3'=1);
[step] w1=21 & stator4=0 -> 1.7999999999999998E-4 : (w1'=4) & (stator4'=1);
[step] w1=21 & stator5=0 -> 8.999999999999999E-5 : (w1'=5) & (stator5'=1);
[step] w1=21 & stator6=0 -> 8.999999999999999E-5 : (w1'=6) & (stator6'=1);
[step] w1=21 & stator13=0 -> 8.999999999999999E-5 : (w1'=13) & (stator13'=1);
[step] w1=21 & stator17=0 -> 8.999999999999999E-6 : (w1'=17) & (stator17'=1);
[step] w1=21 & stator18=0 -> 1.7999999999999998E-4 : (w1'=18) & (stator18'=1);
[step] w1=21 & stator19=0 -> 1.7999999999999998E-4 : (w1'=19) & (stator19'=1);
[step] w1=21 & stator20=0 -> 0.009 : (w1'=20) & (stator20'=1);

[] w1 = 1 ->.000000001 :  (w1'=1);
[] w1 = 2 ->.000000001 :  (w1'=2);
[] w1 = 3 ->.000000001 :  (w1'=3);
[] w1 = 4 ->.000000001 :  (w1'=4);
[] w1 = 5 ->.000000001 :  (w1'=5);
[] w1 = 6 ->.000000001 :  (w1'=6);
[] w1 = 7 ->.000000001 :  (w1'=7);
[] w1 = 8 ->.000000001 :  (w1'=8);
[] w1 = 9 ->.000000001 :  (w1'=9);
[] w1 = 10 ->.000000001 :  (w1'=10);
[] w1 = 11 ->.000000001 :  (w1'=11);
[] w1 = 12 ->.000000001 :  (w1'=12);
[] w1 = 13 ->.000000001 :  (w1'=13);
[] w1 = 14 ->.000000001 :  (w1'=14);
[] w1 = 15 ->.000000001 :  (w1'=15);
[] w1 = 16 ->.000000001 :  (w1'=16);
[] w1 = 17 ->.000000001 :  (w1'=17);
[] w1 = 18 ->.000000001 :  (w1'=18);
[] w1 = 19 ->.000000001 :  (w1'=19);
[] w1 = 20 ->.000000001 :  (w1'=20);
[] w1 = 21 ->.000000001 :  (w1'=21);
endmodule

rewards "steps"
[step] true : 1;
endrewards
```

```
rewards "time"
 true : 1;
endrewards

rewards "blocked" // time spend in blocked anchorages
w1 =2 | w1 =3 | w1 =4 | w1 =5 | w1 =14 | w1 =15 : 1;
endrewards
```

ring.pctl

```
P = ? [F[200*60,200*60] (w1=7)|(w1=17)]
P = ? [F[12000,12000] (w1=7) ]
P = ? [F[12000,12000] (w1=17) ]
R{"steps"}=? [ C<=200*60 ]
R{"blocked"}=? [ C<=200*60 ]
```