Stochastic Simulation of the Kinetics of Multiple Interacting Nucleic Acid Strands

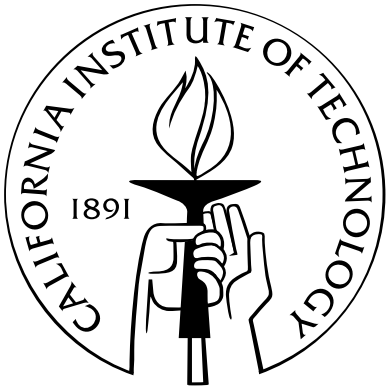
Thesis by

Joseph Malcolm Schaeffer

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2013

(Defended September 28, 2012)

Acknowledgements

Thanks to my advisor Erik Winfree, for his enthusiasm, expertise, and encouragement. The models presented here are due in a large part to helpful discussions with Niles Pierce, Robert Dirks, Justin Bois, and Victor Beck. Two undergraduates, Chris Berlind and Joshua Loving, did summer research projects based on Multistrand and in the process helped build and shape the simulator. There are many people who have used Multistrand and provided very helpful feedback for improving the simulator, especially Josh Bishop, Nadine Dabby, Jonathan Othmer, and Niranjan Srinivas. Nadine Dabby was also invaluable for her feedback and discussions while writing the thesis. Thanks also to the many past and current members of the DNA and Natural Algorithms group for providing a stimulating environment in which to work.

Funding for this work was provided by National Science Foundation grants DMS-0506468 and CCF-0832824, and the Gordon and Betty Moore Foundation through the Caltech Programmable Molecular Technology Initiative.

There are many medical professionals to which I owe my good health while writing this thesis, especially Dr. Jeanette Butler, Dr. Mariel Tourani, Cathy Evaristo, and the staff of the Caltech Health Center, especially Alice, Divina, and Jeannie.

I want to acknowledge all my family and friends for their support. A journey is made all the richer for having good company, and I would not have made it nearly as far without all the encouragement.

Finally, I must thank my wife Lorian, who has been with me every step of this journey and has shared all the high points and low points with her endless love and support.

Abstract

DNA nanotechnology is an emerging field which utilizes the unique structural properties of nucleic acids in order to build nanoscale devices, such as logic gates, motors, walkers, and algorithmic structures. Predicting the structure and interactions of a DNA device requires good modeling of both the thermodynamics and the kinetics of the DNA strands within the system. The kinetics of a set of DNA strands can be modeled as a continuous time Markov process through the state space of all secondary structures. The primary means of exploring the kinetics of a DNA system is by simulating trajectories through the state space and aggregating data over many such trajectories.

We expand on previous work by extending the thermodynamics and kinetics models to handle multiple strands in a fixed volume, and show that the new models are consistent with previous models. We developed data structures and algorithms that allow us to take advantage of local properties of secondary structure, improving the efficiency of the simulator so that we can handle larger systems. The new kinetic parameters in our model were calibrated by analyzing simulator results on experimental systems that measure basic kinetic rates of various processes. Finally, we apply the new simulator to explore a case study on toehold-mediated four-way branch migration.

Contents

Acknowledgements **iii**

Abstract **iv**

List of Figures **viii**

List of Tables **x**

1 Introduction **1**

2 System **4**

2.1 Strands 4

2.2 Complex Microstate 5

2.3 System Microstate 5

3 Energy **7**

3.1 Energy of a System Microstate 7

3.2 Energy of a Complex Microstate 8

3.3 Computational Considerations 10

3.4 Choice of Units 11

4 Kinetics **13**

4.1 Basics 13

4.2 Unimolecular Transitions 15

4.3 Bimolecular Transitions 16

4.4 Transition Rates 17

4.5 Unimolecular Rate Models 18

4.6 Bimolecular Rate Model 19

5 Thermodynamic Equivalence Between the Multistrand and NUPACK Models 21

5.1 Introduction 21

5.2 Definitions 23

5.3 Proof 24

5.3.1 Q_j^{kin} and Q_j 24

5.3.2 Composing Q_{kin} from Q_j^{kin} 27

5.4 Conclusion 30

6 The Simulator: Multistrand 32

6.1 Data Structures 32

6.1.1 Energy Model 32

6.1.2 The Current State: Loop Structure 33

6.1.3 Reachable States: Moves 35

6.2 Algorithms 36

6.2.1 Move Selection 37

6.2.2 Move Update 39

6.2.3 Move Generation 40

6.2.4 Energy Computation 41

6.3 Time Complexity 42

7 Multistrand: Output and Analysis 45

7.1 Trajectory Mode 45

7.1.1 Testing: Energy Model 49

7.1.2 Testing: Kinetics Model 49

7.2 Macrostates 50

7.2.1 Common Macrostates 53

7.3 Transition Mode 55

7.4 First Passage Time Mode 58

7.4.1 Comparing Sequence Designs 61

7.4.2 Systems with Multiple Stop Conditions 63

7.5 Fitting Chemical Reaction Equations 64
7.5.1 Fitting Full Simulation Data to the k_{eff} Model 66
7.6 First Step Mode 67
7.6.1 Fitting the First Step Model 67
7.6.2 Analysis of First Step Model Parameters 68

8 Calibration of Kinetic Parameters **71**
8.1 k_{uni} Calibration 71
8.2 k_{bi} Calibration 73
8.3 Discussion 76
8.3.1 Choice of Experimental Paper for Determining k_{uni} 76
8.3.2 Nonlinearity of k_{bi} Calibrations 77
8.3.3 Other Substrates 78

9 Case Study: Toehold-Mediated Four-Way Branch Migration **79**
9.1 Background 79
9.2 Mechanism 79
9.3 Simulation 81
9.4 Discussion 84

A Strand Orderings for Pseudoknot-Free Representations **86**
A.1 Representation Theorem 90

Bibliography **91**

List of Figures

3.1 Secondary Structure Loop Decomposition 9

4.1 Adjacent Microstate Diagram 14

6.1 Representation of Secondary Structures 34
6.2 Move Data Structure 36
6.3 Move Update Example 39
6.4 Move Generation Example 41
6.5 Full Comparison vs Kinfold 1.0 43

7.1 Trajectory Data 46
7.2 Three-Way Branch Migration System 47
7.3 Trajectory Output after 0.01 s Simulated Time 48
7.4 Trajectory Output after 0.05 s Simulated Time 49
7.5 Example Macrostate 52
7.6 Hairpin Folding Pathways 55
7.7 First Passage Time Data, Design B 60
7.8 First Passage Time Data, Design A 62
7.9 First Passage Time Data, Sequence Design Comparison 62
7.10 First Passage Time Data, 6 Base Toeholds 63
7.11 Starting Complexes and Strand Labels 65
7.12 Final Complexes and Strand Labels 65

8.1 Zippering Mechanism 72

9.1 Four-Way Branch Migration Mechanism 80
9.2 Toehold-Mediated Four-Way Branch Migration Parameterization. 81

9.3 Four-Way Branch Migration Mechanism, Start and Stop States 82
9.4 Bimolecular Success Rate vs Total Toehold Length 84
9.5 Comparison of Experimental and Simulated Rates for Toehold Mediated Four-Way Branch Migration 85

A.1 Polymer Graph Representation 87
A.2 Polymer Graph Changes (Break Move) 88
A.3 Polymer Graph Changes (Join Move) 89

List of Tables

7.1 Two Branch Migration Sequences 48
7.2 Distance Metric Examples 53
7.3 Transition States in Hairpin Pathway 56
7.4 Transition Pathways via Transition Mode Simulation 57
7.5 Transition Pathway Statistics 57
7.6 Transition Pathway Statistics, 100 Trajectories 58
7.7 First Passage Time Data 60

8.1 Average $f(\Delta G_{step})$ for Forward Zippering Steps 72
8.2 Calibrated k_{uni} Parameters 73
8.3 Bimolecular Association Rate k_{hyb} Parameters 74
8.4 Calibrated k_{bi} Parameters 75
8.5 Comparison of Calibrated Parameters 78

9.1 Sequences for Four-Way Branch Migration Domains 83
9.2 Toehold-Mediated Branch Migration, Raw Simulation Results 84

Chapter 1

Introduction

DNA nanotechnology is an emerging field that utilizes the unique structural properties of nucleic acids in order to build nanoscale devices, such as logic gates [23], motors [4, 1], walkers [24, 1, 26], and algorithmic structures [18, 31]. These devices are built out of DNA strands whose sequences have been carefully designed in order to control their secondary structure—the hydrogen bonding state of the bases within the strand (called "base-pairing"). This base-pairing is used to not only control the physical structure of the device, but also to enable specific interactions between different components of the system, such as allowing, for example, a DNA walker to take steps along a prefabricated track. Predicting the structure and interactions of a DNA device requires good modeling of both the thermodynamics and the kinetics of the DNA strands within the system. Thermodynamic models can be used to make equilibrium predictions for these systems, allowing us to look at questions like "Is the walker-track interaction a well-formed and stable molecular structure?", while kinetics models allow us to predict the non-equilibrium dynamics, such as "How quickly will the walker take a step?" While the thermodynamics of multiple interacting DNA strands is a well-studied model [6], which allows for both analysis and design of DNA devices [34, 7], previous work on secondary structure kinetics models only explored the kinetics of how a single strand folds on itself [8].

The kinetics of a set of DNA strands can be modeled as a continuous time Markov process through the state space of all secondary structures. Due to the exponential size of this state space it is computationally intractable to obtain an analytic solution for most problem sizes of interest. Thus the primary means of exploring the kinetics of a DNA system is by simulating trajectories through the state space and aggregating data over many such trajectories. We present here the **Multistrand** kinetics simulator, which extends the previous

work [8] by using the multiple strand thermodynamics model [6] (a core component for calculating transition rates in the kinetics model), adding new terms to the thermodynamics model to account for stochastic modeling considerations, and by adding new kinetic moves that allow bimolecular interactions between strands. Furthermore, we prove that this new kinetics and thermodynamics model is consistent with the prior work on multiple strand thermodynamics models [6].

The **Multistrand** simulator is based on the Gillespie algorithm [9] for generating statistically correct trajectories of a stochastic Markov process. We developed data structures and algorithms that take advantage of local properties of secondary structures. These algorithms enable the efficient reuse of the basic objects that form the system, such that only a very small part of the state's neighborhood information needs to be recalculated with every step. A key addition was the implementation of algorithms to handle the new kinetic steps that occur between different DNA strands, without increasing the time complexity of the overall simulation. These improvements lead to a reduction in worst case time complexity of a single step and also led to additional improvements in the average case time complexity.

What data does the simulation produce? At the very simplest, the simulation produces a full kinetic trajectory through the state space—the exact states it passed through, and the time at which it reached them. A small system might produce trajectories that pass through hundreds of thousands of states, and that number increases rapidly as the system gets larger. Going back to our original question, the type of information a researcher hopes to get out of the data could be very simple: "How quickly does the walker take a step?", with the implied question of whether it's worth it to actually purchase the particular DNA strands composing the walker to perform an experiment, or go back to the drawing board and redesign the device. One way to acquire that type of information is to look at the first time in the trajectory where we reached the "walker took a step" state, and record that information for a large number of simulated trajectories in order to obtain a useful answer. We designed and implemented new simulation modes that allow the full trajectory data to be condensed as it's generated into only the pieces the user cares about for their particular question. This analysis tool also required the development of flexible ways to talk about states that occur in trajectory data; if someone wants data on when the walker took a step, we have to be able to express that in terms of the Markov process states which meet that condition.

Chapters 1, 2, 3, 4, 6, 7, and Appendix A originally appeared in my Master's thesis. Chapter 5 is a completely new proof of equivalence between the thermodynamics model we develop in Chapter 3 and the NUPACK model. Chapter 8 discusses how we calibrate the kinetics parameters k_{uni} and k_{bi} which were introduced in Chapter 4. Chapter 9 is a case study on using the simulator to explore a toehold-mediated four-way branch migration. The experimental data k_1^{fit} in Chapter 9 is from Dabby, et al. [5] on which I am a co-author. I performed simulations using Multistrand for that work, which appear in that paper and are presented in more detail here.

Found in the Master's thesis but not here are two appendices which describe the software design for the data structures and algorithms used in simulator, as well as a different proof of equivalence between our thermodynamics model and the NUPACK model.

Chapter 2

System

We are interested in simulating nucleic acid molecules (DNA or RNA) in a stochastic regime; that is to say that we have a discrete number of molecules in a fixed volume. This regime is found in experimental systems that have a small volume with a fixed count of each molecule present, such as the interior of a cell. We can also apply this to experimental systems with a larger volume (such as a test tube) when the system is well mixed, as we can pick a fixed (small) volume and deal with the expected counts of each molecule within it, rather than the whole test tube.

To discuss the modeling and simulation of the system, we need to be very careful to define the components of the system, and what comprises a state of the system within the simulation.

2.1 Strands

Each DNA molecule to be simulated is represented by a *strand*. Our system then contains a set of strands Ψ^*, where each strand $s \in \Psi^*$ is defined by $s = (id, label, sequence)$. A strand's *id* uniquely identifies the strand within the system, while the *sequence* is the ordered list of nucleotides that compose the strand.

Two strands could be considered *identical* if they have the same sequence. However, in some cases it is convenient to make a distinction between strands with identical sequences. For example, if one strand were to be labeled with a fluorophore, it would no longer be physically identical to another with the same sequence but no fluorophore. Thus, the *label* is used to designate whether two strands are identical. We define two strands as being *identical* if they have the same labels and sequences. In most cases this distinction between

the label and the sequence is not used, so it will be explicitly noted when it is important.

2.2 Complex Microstate

A *complex* is a set of strands connected by base pairing (secondary structure). We define the state of a complex by $c = (ST, \pi^*, BP)$, called the "complex microstate". The components are a nonempty set of strands $ST \subseteq \Psi^*$, an ordering π^* on the strands ST, and a list of base pairings $BP = \{(i_j \cdot k_l) \mid$ base i on strand j is paired to base k on strand l, and $j \leq l$, with $i < k$ if $j = l\}$, where we note that "strand l" refers to the strand occurring in position l in the ordering π^*. Note that we require a complex to be "connected": there is no proper subset of strands in the complex for which the base pairings involving those strands do not involve at least one base outside that subset. Given a complex microstate c, we will use $ST(c), \pi^*(c), BP(c)$ to refer to the individual components.

While this definition defines the full space of complex microstates, it is common to disallow some secondary structures due to physical or computational constraints. For example, we disallow the pairing of a base with any other within three bases on the same strand, as this would correspond to an impossible physical configuration. Another class of disallowed structures are called the *pseudoknotted* secondary structures, which require computationally difficult energy model calculations, and are fully defined and discussed further in Appendix A.

2.3 System Microstate

A system microstate represents the configuration of the strands in the volume we are simulating (the "box"). Since we allow complexes to be formed of one or more strands, every unique strand in the system must be present in a single complex and thus we can represent the system microstate by a set of those complexes.

We define a *system microstate* i as a set of complex microstates, such that each strand in the system is in exactly one complex within the system. This is formally stated in the following equation:

$$\bigcup_{c \in i} ST(c) = \Psi^* \text{ and } \forall c, c' \in i \text{ with } c \neq c', ST(c) \cap ST(c') = \emptyset \tag{2.1}$$

This definition leads to the natural use of $|i|$ to indicate the number of complexes present in system microstate i, and $i \setminus j$ to indicate the complex microstates present in system microstate i that are not in j.

Chapter 3

Energy

The conformation of a nucleic acid strand at equilibrium can be predicted by a well-studied model, called the nearest neighbor energy model [20, 19, 21]. Recent work has extended this model to cover systems with multiple interacting nucleic acid strands [6].

The distribution of system microstates at equilibrium is a Boltzmann distribution, where the probability of observing a microstate i is given by

$$Pr(i) = \frac{1}{Q_{kin}} * e^{-\Delta G_{box}(i)/RT} \tag{3.1}$$

where $\Delta G_{box}(i)$ is the free energy of the system microstate i, and is the key quantity determined by these energy models. $Q_{kin} = \sum_i e^{-\Delta G_{box}(i)/RT}$ is the partition function of the system, R is the gas constant, and T is the temperature of the system.

3.1 Energy of a System Microstate

We now consider the energy of the system microstate i, and break it down into components. The system consists of many complex microstates c, each with their own energy. We also must account for the entropy of the system (the number of configurations of the complexes spatially within the "box") in the energy, and thus must define these two terms.

Let us first consider the entropy term. We consider the "zero" energy system microstate to be the one in which all strands are in separate complexes, thus our entropy term is in terms of the reduction of available states caused by having strands join together.

We assume that the number of complexes in the system, C, is much smaller than the number of states within our box, $\frac{V}{V_0}$, where V is the simulated volume, and V_0 is our reference volume[1] chosen to be consistent with existing thermodynamic models (see Sections 3.4 and 5).

We can then approximate the standard statistical entropy of the system as $C*RT\log\frac{V}{V_0}$. Letting L_{tot} be the total number of strands in the system, our zero state is then $L_{tot}*RT\log\frac{V}{V_0}$. Defining $\Delta G_{volume} = RT\log\frac{V}{V_0}$, the contribution to the energy of the system microstate i from the entropy of the box is then:

$$(L_{tot} - C) * \Delta G_{volume}$$

And thus in terms of $C, L_{tot}, \Delta G_{volume}$ and $\overline{\Delta G}(c)$ (the energy of complex microstate c, defined in the next section), we define $\Delta G_{box}(i)$, the energy of the system microstate i, as follows:

$$\Delta G_{box}(i) = (L_{tot} - C) * \Delta G_{volume} + \sum_{c \in i} \overline{\Delta G}(c)$$

The energy formulas derived here, suitable for our stochastic model, differ from those in [6] in two main ways: the lack of symmetry terms, and the addition of the ΔG_{volume} term. We compare this stochastic model to the mass action model in much more detail in Section 5.

3.2 Energy of a Complex Microstate

We previously defined a complex microstate in terms of the list of base pairings present within it. However, the well-studied models are based upon nearest neighbor interactions between the nucleic acid bases. These interactions divide the secondary structure of the

[1] We calculate V_0 as the volume in which we would have exactly one molecule at a standard concentration of 1 mol/L: $V_0 = 1/(N_a * 1\,\text{mol/L})$, where N_a is Avogadro's number, and thus V_0 is in liters.

Similarly, we may wish to calculate V based on the concentration u in mol/L of a single strand such that the volume V is chosen such that exactly one molecule is present in that volume. In this case we have $V = \frac{1}{u*N_a}$ and our number of states in the box is then $\frac{V}{V_0} = \frac{N_a}{u*N_a} = \frac{1}{u}$.

system into local components which we refer to as *loops*, shown in Figure 3.1.

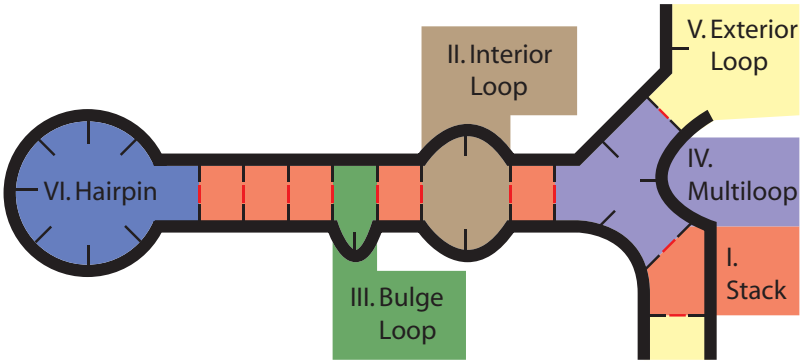


Figure 3.1: Secondary structure divided into loops

These loops can be broken down into different categories, and parameter tables and formulas for each category have been determined from experimental data [21]. Each loop l has an energy, $\Delta G(l)$ which can be retrieved from the appropriate parameter table for its category. Each complex also has an energy contribution associated with the entropic initiation cost [3] (e.g., rotational) of bringing two strands together, ΔG_{assoc}, and the total contribution is proportional to the number of strands L within the complex, as follows: $(L-1) * \Delta G_{assoc}$.

The energy of a complex microstate c is then the sum of these two types of contributions. We can also divide any free energy ΔG into the enthalpic and entropic components, ΔH and ΔS related by $\Delta G = \Delta H + T * \Delta S$, where T is the temperature of the system. For a complex microstate, each loop can have both enthalpic and entropic components, but ΔG_{assoc} is usually assumed to be purely entropic [20]. This becomes important when determining the kinetic rates, in Section 4.

We use $\overline{\Delta G}(c)$ to refer to the energy of a complex microstate to be consistent with the nomenclature in [6], where $\overline{\Delta G}(c)$ refers to the energy of a complex when all strands within it are consider unique (as is the case in our system), and $\Delta G(c)$ is the energy of the complex, without assuming that all strands are unique (and thus it must account for rotational symmetries). This is discussed more in Section 5.

In summary, the standard free energy of a complex microstate c, containing $L = |ST(c)|$ strands:

$$\overline{\Delta G}(c) = \left(\sum_{\text{loop } l \in c} \Delta G(l) \right) + (L-1)\Delta G_{assoc}$$

3.3 Computational Considerations

While the simulator could use the system microstate energy in the form given in the previous sections, it is convenient for us to group terms such that the computation need only take place per complex. Thus we wish to include the $(L_{tot} - C)\Delta G_{volume}$ term in the energy computation for the complex microstates. Recall that L_{tot} is the number of strands in the system, and C is the number of complexes in the system microstate. Assuming that we are computing the energy ΔG_{box} of system microstate i, we can rewrite L_{tot} and C as follows:

$$\begin{aligned} L_{tot} &= \sum_{c \in i} |ST(c)| \\ C &= \sum_{c \in i} 1 \end{aligned}$$

And thus arrive at:

$$\Delta G_{box}(i) = \sum_{c \in i} \left(\overline{\Delta G}(c) + (|ST(c)| - 1) * \Delta G_{volume} \right)$$

We then define $\Delta G^*(c) = \overline{\Delta G}(c) + (|ST(c)| - 1) * \Delta G_{volume}$, and $L(c) = |ST(c)|$ and thus have the following forms for the energy of a system microstate and the energy of a complex microstate:

$$\begin{aligned}\Delta G_{box}(i) &= \sum_{c \in i} \Delta G^*(c) \\ \Delta G^*(c) &= \left(\sum_{\text{loop } l \in c} \Delta G(l) \right) + (L(c) - 1) * (\Delta G_{assoc} + \Delta G_{volume})\end{aligned}$$

3.4 Choice of Units

The free energy ΔG° for a reaction $A + B \rightleftharpoons C$ is usually expressed in terms of the equilibrium constant K_{eq} and the concentrations $[A], [B], [C]$ (in mol/L) of the molecules involved, as follows: $e^{\Delta G^\circ / RT} = K_{eq} = \frac{[A][B]}{[C]}$. We can also express the free energy $\Delta G'$ in terms of the dimensionless mole fractions x_A, x_B, x_C, where $x_i = [i]/\rho_{H_2O}$ (for dilute solutions), and ρ_{H_2O} is the molarity of water (55.14 mol/L at 37 °C). In this case, we have $e^{\Delta G'/RT} = K'_{eq} = \frac{x_A * x_B}{x_C}$, and relating it to the previous equation, we see that $e^{\Delta G'/RT} = \frac{([A]/\rho_{H_2O}) * ([B]/\rho_{H_2O})}{[C] * \rho_{H_2O}} = \frac{[A][B]}{[C]} * \frac{1}{\rho_{H_2O}} = e^{\Delta G^\circ / RT} * e^{-\log \rho_{H_2O}}$. Thus if we have an energy ΔG° which was for concentration units and we wish to use mole fraction units, we must adjust it by $-RT \log \rho_{H_2O}$ (-2.47 kcal/mol at 37 °C) to obtain the correct quantity. In general, if we have a complex of N molecules, the conversion to mole fractions will require an adjustment of $-(N-1) * RT \log \rho_{H_2O}$.

In the reference [6], free energies are always in mole fraction units, and the $-RT \log \rho_{H_2O}$ term is included as part of the ΔG^{assoc} term (footnote 13 in [6]), as follows: $\Delta G^{assoc} = \Delta G^{pub}_{assoc} - RT \log \rho_{H_2O}$, where ΔG^{pub}_{assoc} is the reference value found in [3] (1.96 kcal/mol at 37 °C) and is the value we use for ΔG_{assoc} when using molar concentration units.

Since we expect the probability of observing a particular complex microstate to remain the same no matter what reference units we use for the free energy, this implies that if we wanted to express our $\Delta G^*(c)$ in mole fraction units, we would need to use $\Delta G_{assoc} = \Delta G^{pub}_{assoc} - RT \log \rho_{H_2O}$ and $\Delta G_{volume} = RT \log \frac{V}{V_0} + RT \log \rho_{H_2O}$ in order for the probability of observing the state c to remain the same. We note that this ΔG_{volume} simplifies to being $\Delta G_{volume} = RT \log(V * \rho_{H_2O} * N_a) = RT \log M_s$, where M_s is the number of solvent molecules found in the volume V.

We use the molar concentration units ($\Delta G_{volume} = RT \log \frac{V}{V_0} = RT \log \frac{1}{u}$, $\Delta G_{assoc} = \Delta G^{pub}_{assoc}$) for the remainder of this thesis, except in Section 5 where we use mole fraction

units ($\Delta G_{volume} = RT \log M_s$, $\Delta G_{assoc} = \Delta G^{pub}_{assoc} - RT \log \rho_{H_2O}$) to be consistent with the units in [6].

Chapter 4

Kinetics

4.1 Basics

Thermodynamic predictions have only limited use for some systems of interest, if the key information to be gathered is the reaction rates and not the equilibrium states. Many systems have well-defined ending states that can be found by thermodynamic prediction, but predicting whether it will reach the end state in a reasonable amount of time requires modeling the kinetics. Kinetic analysis can also help uncover poor sequence designs, such as those with alternate reactions leading to the same states, or kinetic traps which prevent an intended reaction from occurring quickly.

The kinetics are modeled as a continuous time Markov process over secondary structure space. System microstates i, j are considered adjacent if they differ by a single base pair (Figure 4.1), and we choose the transition rates k_{ij} (the transition from state i to state j) and k_{ji} such that they obey detailed balance:

$$\frac{k_{ij}}{k_{ji}} = e^{-\frac{\Delta G_{box}(j) - \Delta G_{box}(i)}{RT}} \tag{4.1}$$

This property ensures that given sufficient time we will arrive at the same equilibrium state distribution as the thermodynamic prediction, (i.e., the Boltzmann distribution on system microstates, equation 3.1) but it does not fully define the kinetics as this only constrains the ratio $\frac{k_{ij}}{k_{ji}}$. We discuss how to choose these transition rates in the following sections, but regardless of this choice, we can still determine how the next state is chosen and the time at which that transition occurs.

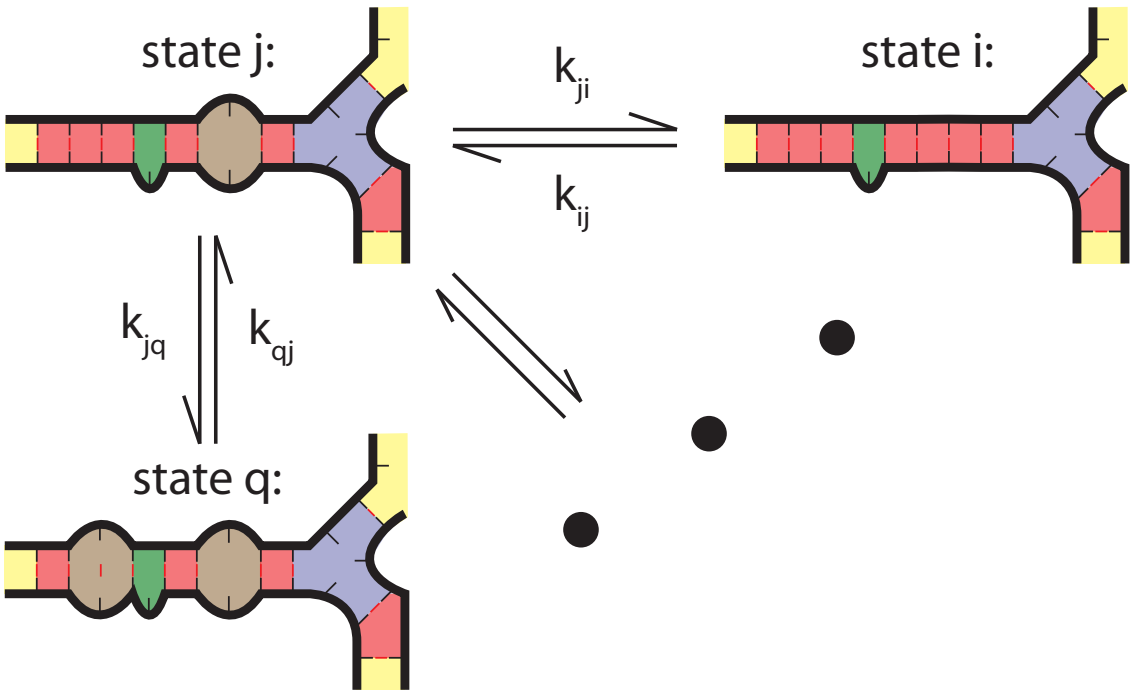


Figure 4.1: System microstates i, q adjacent to current state j, with many others not shown

Given that we are currently in state i, the next state m in a simulated trajectory is chosen randomly among the adjacent states j, weighted by the rate of transition to each.

$$Pr(m) = \frac{k_{im}}{\Sigma_j k_{ij}} \tag{4.2}$$

Similarly, the time taken to transition to the next state is chosen randomly from an exponential distribution with rate parameter λ, where λ is the total rate out of the current state, $\Sigma_j k_{ij}$.

$$Pr(\Delta t) = \lambda \exp(-\lambda \Delta t) \tag{4.3}$$

We will now classify transitions into two exclusive types: those that change the number of complexes present in the system, called *bimolecular transitions*, and those where changes are within a single complex, called *unimolecular transitions*.

4.2 Unimolecular Transitions

Because unimolecular transitions involve only a single complex, it is natural to define these transitions in terms of the complex microstate which changed, rather than the full system microstate. Like Figure 4.1 implies, we define a complex microstate d as being adjacent to a complex microstate c if it differs by exactly one base pair. We call a transition from c to d that adds a base pair a *creation* move, and a transition from c to d that removes a base pair a *deletion* move. The exclusion of pseudoknotted structures is not inherent in this definition of adjacent states, but rather arises from our disallowing pseudoknotted complex microstates.

In more formal terms we now define the adjacent states to a system microstate, rather than those adjacent to a complex microstate as in the simple definition above. Recall from Section 2.3 that $|i|$ is the number of complexes present in system microstate i, and $i \setminus j$ is the set of complex microstates in i that are not also in system microstate j.

Two system microstates i, j are adjacent by a unimolecular transition iff $\exists c \in i, d \in j$ such that:

$$|i| = |j| \text{ and } i \setminus j = \{c\} \text{ and } j \setminus i = \{d\} \tag{4.4}$$

and one of these two holds:

$$BP(c) \subset BP(d) \text{ and } |BP(d)| = |BP(c)| + 1 \tag{4.5}$$

$$BP(d) \subset BP(c) \text{ and } |BP(c)| = |BP(d)| + 1 \tag{4.6}$$

In other words, the only differences between i and j are in c and d, and they differ by exactly one base pair. If equation 4.5 is true, we call the transition from i to j a *base pair creation move*, and if equation 4.6 is true, we call the transition from i to j a *base pair deletion move*. Note that if i to j is a creation move, j to i must be a deletion move, and vice versa. Similarly, if there is no transition from i to j, there cannot be a transition from j to i, which implies that every unimolecular move in this system is reversible.

4.3 Bimolecular Transitions

A bimolecular transition from system microstate i to system microstate j is one where the single base pair difference between them leads to a differing number of complexes within each system microstate. This differing number of complexes could be due to a base pair joining two complexes in i to form a single complex in j, which we will call a *join move*. Conversely, the removal of this base pair from i could cause one complex in i to break into two complexes within j, which we will call a *break move*. Note that if i to j is a join move, then j to i must be a break move, and vice versa. As we saw before, this also implies that every bimolecular move is reversible.

Formally, a transition from system microstate i to system microstate j is a join move if $|i| = |j| + 1$ and we can find complex microstates $c, c' \in i$ and $d \in j$, with $c \neq c'$ such that the following equations hold:

$$i \setminus \{c, c'\} = j \setminus \{d\} \tag{4.7}$$

$$\exists x \in BP(d) \text{ s.t. } BP(d) \setminus \{x\} = BP(c) \cup BP(c') \tag{4.8}$$

Similarly, a transition from system microstate i to system microstate j is a break move if $|i| + 1 = |j|$ and we can find complex microstates $c \in i$ and $d, d' \in j$ with $d \neq d'$ such that the following equations hold:

$$i \setminus \{c\} = j \setminus \{d, d'\} \tag{4.9}$$

$$\exists x \in BP(c) \text{ s.t. } BP(c) \setminus \{x\} = BP(d) \cup BP(d') \tag{4.10}$$

While arbitrary bimolecular transitions are not inherently prevented from forming pseudoknots in this model, we again implicitly prevent them by using only complex microstates that are not pseudoknotted.

4.4 Transition Rates

Now that we have defined all of the possible transitions between system microstates, we must decide how to assign rates to each transition. We know that if there is a transition from system microstate i to system microstate j with rate k_{ij} there must be a transition from j to i with rate k_{ji} which are related by:

$$\frac{k_{ij}}{k_{ji}} = e^{-\frac{\Delta G_{box}(j) - \Delta G_{box}(i)}{RT}} \tag{4.11}$$

This condition is known as *detailed balance*, and does not completely define the rates k_{ij}, k_{ji}. Thus a key part of our model is the choice of *rate method*, the way we set the rates of a pair of reactions so that they obey detailed balance.

While our simulator can use any arbitrary rate method we can describe, we would like our choice to be physically realistic (i.e., accurate and predictive for experimental systems). There are several rate methods found in the literature [11, 12, 36] which have been used for kinetics models for single-stranded nucleic acids [8, 36] with various energy models. As a start, we have implemented three of these simple rate methods which were previously used in single base pair elementary step kinetics models for single stranded systems. In addition we present a rate method for use in bimolecular transitions that is physically consistent with both mass action and stochastic chemical kinetics. We verify that the kinetics model (and thus our choice of rate method) have been correctly implemented by verifying that the detailed balance condition holds (Section 7.1.2).

In order to maintain consistency with known thermodynamic models, each pair of k_{ij} and k_{ji} must satisfy detailed balance and thus their ratio is determined by the thermodynamic model, but in principle each pair could be independently scaled by some arbitrary prefactor, perhaps chosen to optimize agreement with experimental results on nucleic acid kinetics. However, since the number of microstates is exponential, this leads to far more model parameters (the prefactors) than is warranted by available experimental data. For the time being, we limit ourselves to using only two scaling factors: k_{uni} for use with unimolecular transitions, and k_{bi} for bimolecular transitions.

4.5 Unimolecular Rate Models

The first rate model we will examine is the Kawasaki method [11]. This model has the property that both "downhill" (energetically favorable) and uphill transitions scale directly with the steepness of their slopes.

$$k_{ij} = k_{uni} * e^{-\frac{\Delta G_{box}(j) - \Delta G_{box}(i)}{2RT}} \tag{4.12}$$

$$k_{ji} = k_{uni} * e^{-\frac{\Delta G_{box}(i) - \Delta G_{box}(j)}{2RT}} \tag{4.13}$$

The second rate model under consideration is the Metropolis method [12]. In this model, all downhill moves occur at the same fixed rate, and only the uphill moves scale with the slope. This means that the maximum rate for any move is bounded, and in fact all downhill moves occur at this rate. This is in direct contrast to the Kawasaki method, where there is no bound on the maximum rate.

$$\text{if } \Delta G_{box}(i) > \Delta G_{box}(j) \text{ then} \quad k_{ij} = 1 * k_{uni} \tag{4.14}$$

$$k_{ji} = k_{uni} * e^{-\frac{\Delta G_{box}(i) - \Delta G_{box}(j)}{RT}} \tag{4.15}$$

$$\text{otherwise,} \quad k_{ij} = k_{uni} * e^{-\frac{\Delta G_{box}(j) - \Delta G_{box}(i)}{RT}} \tag{4.16}$$

$$k_{ji} = 1 * k_{uni} \tag{4.17}$$

Finally, the entropy/enthalpy method [36] uses the division of free energies into entropic and enthalpic components to assign the transition rates in an intuitive manner: base pair creation moves must overcome the entropic energy barrier to bring the bases into contact, and base pair deletion moves must overcome the enthalpic energy barrier in order to break them apart.

$$\text{if } i \text{ to } j \text{ is a creation:} \quad k_{ij} = k_{uni} * e^{\frac{\Delta S_{box}(j) - \Delta S_{box}(i)}{R}} \tag{4.18}$$

$$k_{ji} = k_{uni} * e^{-\frac{\Delta H_{box}(i) - \Delta H_{box}(j)}{RT}} \tag{4.19}$$

$$\text{otherwise,} \quad k_{ij} = k_{uni} * e^{-\frac{\Delta H_{box}(j) - \Delta H_{box}(i)}{RT}} \tag{4.20}$$

$$k_{ji} = k_{uni} * e^{\frac{\Delta S_{box}(i) - \Delta S_{box}(j)}{R}} \tag{4.21}$$

We note that the value of k_{uni} that best fits experimental data is likely to be different for all three models (see Section 8). Additionally, due to equations 4.5 and 4.6, the energies of system microstates i and j, $\Delta G_{box}(i)$ and $\Delta G_{box}(j)$ differ in exactly one pair of complex microstates $c \in i, d \in j$, and by exactly three loop terms in those complex microstates.

4.6 Bimolecular Rate Model

When dealing with moves that join or break complexes, we must consider the choice of how to assign rates for each transition in a new light. In the particular situation of the join move, where two molecules in a stochastic regime collide and form a base pair, this rate is expected to be modeled by stochastic chemical kinetics.

Stochastic chemical kinetics theory [9] tells us that there should be a rate constant k such that the propensity of a particular bimolecular reaction between two species X and Y should be $k * \#X * \#Y/V$, where $\#X$ and $\#Y$ are the number of copies of X and Y in the volume V. Since our simulation considers each strand to be unique, $\#X = \#Y = 1$, and thus we see the propensity should scale as $1/V$. Recalling that $\Delta G_{volume} = RT \log \frac{V}{V_0} = RT \log \frac{1}{u}$, we see that we can obtain the $1/V$ scaling by letting the join rate be proportional to $e^{-\Delta G_{volume}/RT}$.

Thus, we arrive at the following rate method, and note that the choice of k (above) or our scalar term k_{bi} can be found by comparison to experiments measuring the hybridization rate of oligonucleotides [28] (see Section 8).

$$\text{if } i \text{ to } j \text{ is a complex join move:} \quad k_{ij} = k_{bi} * e^{\frac{-\Delta G_{volume}}{RT}} = k_{bi} * \frac{V_0}{V} = k_{bi} * u \tag{4.22}$$

$$k_{ji} = k_{bi} * e^{-\frac{\Delta G_{box}(i) - \Delta G_{box}(j) + \Delta G_{volume}}{RT}} \tag{4.23}$$

$$\text{otherwise,} \quad k_{ij} = k_{bi} * e^{-\frac{\Delta G_{box}(j) - \Delta G_{box}(i) + \Delta G_{volume}}{RT}} \tag{4.24}$$

$$k_{ji} = k_{bi} * e^{-\frac{\Delta G_{volume}}{RT}} \tag{4.25}$$

We note that like the bimolecular case, equations 4.7–4.10 imply that the system microstates i and j differ by exactly three loop terms in their complex microstates. However, they also differ in the total number of complexes within each system microstate, such that if i to j is a join move, $\Delta G_{box}(i) - \Delta G_{box}(j) = \Delta G_{loops}(i,j) - \Delta G_{volume} - \Delta G_{assoc}$, where $\Delta G_{loops}(i,j)$ represents the energy differences between i and j due to the three differing loop terms in the complex microstates.

This formulation is convenient for simulation, as the join rates are then independent of the resulting secondary structure. We could use the other choices for assigning rates from 4.4, but they would require much more computation time. While the above model is of course an approximation to the physical reality (albeit one which we believe at least intuitively agrees with what we expect from stochastic chemical kinetics), if we later determine there is a better approximation we could use that instead, even if it cost us a bit in computation time. One issue in the above model that we wish to revisit in the future is that due to the rate being determined for **every** possible first base pair between two complexes, the overall rate for two complexes to bind (by a single base pair) is proportional roughly to the square of the number of exposed nucleotides (although possibly only a linear subset is likely to zipper up reliably), in addition to the $\frac{1}{V}$ dependence noted earlier.

Chapter 5

Thermodynamic Equivalence Between the Multistrand and NUPACK Models

5.1 Introduction

We now wish to compare the Multistrand thermodynamics model and that of NUPACK [34], which is described in [6]. These two models are over very similar state spaces; in the Multistrand model we look at system microstates of a fixed volume where all strands are uniquely labeled, and in the NUPACK model the system has states in a fixed volume, but the strands in the system are not necessarily unique. So a natural way to begin comparing the two models is to look at the probability of "events" in the system, such as the probability of observing a particular state in the NUPACK model and the equivalent system microstates in the Multistrand model after the system has been run long enough to reach equilibrium. In order to calculate these probabilities, the quantity of interest is the partition function for these models.

Let us consider the partition function for our system:

$$Q_{kin} = \sum_{s \in \chi} e^{-\Delta G(s)/RT}$$

where χ is the set of all possible unpseudoknotted system microstates for our fixed set of unique strands in a box of volume V containing M_s solvent molecules. We would like to show how this partition function Q_{kin} relates to the partition function Q_{box} used in NUPACK for the thermodynamics of multiple strand systems [6]. The partition function

Q_{box} is for a system where strands are considered identical if they have the same sequences and thus have symmetry factors when computing the energy of a complex. Additionally, Q_{box} accounts for the energetics of the "box" (the volume in which the strands are present) as part of the partition function itself, rather than in the energy of system states as is done in our kinetic system.

In the Multistrand thermodynamics model, strands in the system are defined by three pieces of information: a unique identifier, a label, and a sequence. In all of our previous analysis, strands are always considered unique. However, in order to compare our system with that of Dirks, et al. [6], we allow for the possibility of strands being considered *indistinguishable* if they share the same label and sequence. This will allow us to establish an equivalence between microstates in our model and those found in the thermodynamics work. We call the combination of label and sequence a *strand type*, thus if two strands are the same type, they are considered indistinguishable.

The thermodynamic partition function Q_{box} is expressed (at the lowest level) in terms of partition functions Q_j, which is the partition function over a single complex with type j independent of volume. A complex type is a fixed set of strand types and counts of each strand type, and represents a connected complex containing those strands (regardless of their unique id), for example, a complex type might be $\{3 * A, 2 * B\}$, where A and B are strand types. Thus the partition function over this complex type is the partition function over all possible states of a single connected complex containing three A strands and two B strands in any order. We now wish to break down our partition function Q_{kin} into smaller components in a similar way.

One way to approach the division of Q_{kin} into smaller components is to look at the complex partition functions for a particular set of strands ids. For example, in a system with three strands (regardless of their types), one way to write the partition function is as follows:

$$Q_{kin} = \overline{Q}_{\{1\}} * \overline{Q}_{\{2\}} * \overline{Q}_{\{3\}} + \overline{Q}_{\{1,2\}} * \overline{Q}_{\{3\}} + \overline{Q}_{\{1,3\}} * \overline{Q}_{\{2\}} + \overline{Q}_{\{2,3\}} * \overline{Q}_{\{1\}} + \overline{Q}_{\{1,2,3\}}$$

where $\overline{Q}_r$ is the partition function over all complex microstates which involve the set r of unique strands. Continuing this example, we let our three strands $\{1, 2, 3\}$ (where numbers indicate their unique ids) have types $\{A, A, B\}$, respectively. Now let Q_j^{kin} be the partition function over complex microstates of a complex type j which used a fixed set of unique identifiers—e.g., $Q_{\{A\}}^{kin}$ is the partition function over all complex microstates which involve a single strand of type A, with a fixed identifier. Note that we distinguish this with the superscript *kin* due to the canonical Q_j being over complex states where strands may be considered indistinguishable, as opposed to our complex microstates where all strands are unique. So now $\overline{Q}_{\{1\}} = \overline{Q}_{\{2\}} = Q_{\{A\}}^{kin}$, and so on, so we can simplify our original breakdown of the partition function as follows:

$$Q_{kin} = \left(Q_{\{A\}}^{kin}\right)^2 * Q_{\{B\}}^{kin} + Q_{\{2*A\}}^{kin} * Q_{\{B\}}^{kin} + 2 * Q_{\{A,B\}}^{kin} * Q_{\{A\}}^{kin} + Q_{\{2*A,B\}}^{kin}$$

We now observe that two of the original terms in the expression for Q_{kin} have been collected into the single term $2 * Q_{\{A,B\}}^{kin} * Q_{\{A\}}^{kin}$. We could calculate the coefficient in front of the $Q_{\{A,B\}}^{kin} * Q_{\{A\}}^{kin}$ term by determining how many distinct ways we could assign unique strand identifiers to the A and B strands that would lead to complexes consistent with those complex types. In this case there are only two ways: the only strand of type B (with unique id 3) could be paired with either one of the two strands of type A (with unique ids $1, 2$) to form the $\{A, B\}$ complex, and then the other strand is in the single $\{A\}$ complex.

In this proof, we will show how to relate the Q_j^{kin} to the canonical Q_j, as well as how to calculate the coefficient present in our Q_{kin} summation for each term when using the Q_j^{kin} form, which amounts to counting the number of ways we could assign the unique strand ids in such a way as to match the complex types present in each term.

5.2 Definitions

We previously introduced the set of (uniquely labeled) strands Ψ^*, let us now introduce the equivalent for the set of strand types. Ψ^0 is the set of strand types (also known as species) in our system, e.g., for our example system above, $\Psi^0 = \{A, B\}$. We also have the set of possible complex types in our system, Ψ, where each complex type (as mentioned above) is

a set of strand types and the number of each present in the connected complex. We note that since our set of strands is finite, the set of complex types is also finite though it can be quite large.

To keep track of which complex types are present in a system microstate, we introduce the population vector $m \in \mathbb{N}^{|\Psi|}$, where we note $\mathbb{N} = \mathbb{Z}_{\geq 0}$. The population vector indicates the number of complexes of that type present in the system microstate. The initial population vector $m^0 \in \mathbb{N}^{|\Psi^0|}$ indicates how many of each type of strand are present in the system. We relate the two by the strand matrix $A \in \mathbb{N}^{|\Psi^0| \times |\Psi|}$, whose entries A_{ij} correspond to the number of strands of strand type i in complex type j. By our previous definition of strand complex, the columns of A are distinct because each column specifies the number of strands of each type present in the complex type j for that column. Thus, if we have a system which starts with m^0 of each strand species, $\Lambda = \{m \,|\, Am = m^0\}$ is the set of all population vectors consistent with conservation of strand counts.

Since [6] uses mole fraction units for all energies (see discussion in Section 3.4), we use the same units for ΔG_{volume} and ΔG_{assoc}, thus we have:

$$\begin{aligned} \Delta G_{volume} &= RT \log M_s \\ \Delta G_{assoc} &= \Delta G^{pub}_{assoc} - RT \log \rho_{H_2O} = \Delta G^{assoc} \end{aligned}$$

where ΔG^{assoc} is the equivalent term used in [6].

5.3 Proof

5.3.1 Q_j^{kin} and Q_j

We know from [6] that Q_j is defined as:

$$Q_j = \sum_{\pi \in \Pi_j} Q_j(\pi)$$

where Π_j is the set of distinguishable orderings on the strand types present in complex type j. E.g., if j is a complex type with $\{3*A, 2*B\}$, Π_j is $\{(A,A,A,B,B),(A,A,B,A,B)\}$ as every other ordering is equivalent to one of these two via a cyclic permutation (a permutation on objects such that the ordering would remain the same when laid out on a circle). $Q_j(\pi)$ is then the partition function over all states $\Omega_j(\pi)$ which have a particular indistinguishable ordering π, as follows:

$$Q_j(\pi) = \sum_{c \in \Omega_j(\pi)} \exp(-\Delta G(c)/RT)$$

In this summation, we use the free energy of a complex state with (possibly) indistinguishable strands, and thus there is an extra energy term relating to the rotational symmetry $R(c)$, as described in [6] as the factor R (we distinguish here due to different units leading to the prefactor being RT rather than kT). This is related to our $\overline{\Delta G}(c)$ described in Section 3.2 as follows:

$$\Delta G(c) = RT \log R(c) + \overline{\Delta G}(c) \tag{5.1}$$

We now wish to examine Q_j^{kin} and see how it relates to Q_j.

$$Q_j^{kin} = \sum_{\pi \in \overline{\Pi}_j} \sum_{c \in \overline{\Omega}_j(\pi)} \exp(-\Delta G^*(c)/RT) \tag{5.2}$$

where $\overline{\Pi}_j$ is the set of circular orderings on the strand identifiers present in complex type j, $\overline{\Omega}_j(\pi)$ is the set of all complex microstates which have ordering π, the complex microstate energy $\Delta G^*(c) = \overline{\Delta G}(c) + (L(c)-1) * \Delta G_{volume}$, and where $L(c)$ is the total number of strands in c.

For example, if our complex type j is $\{3*A, 2*B\}$ like before, and we are given the ids $\{1,2,3\}$ as strand type A, and $\{4,5\}$ as strand type B, then we have:

$$\begin{aligned}\overline{\Pi}_j = \{&\{1,2,3,4,5\}, \quad \{1,2,3,5,4\}, \quad \{1,2,5,3,4\}, \quad \{1,2,5,4,3\},\\ &\{1,2,4,3,5\}, \quad \{1,2,4,5,3\}, \quad \{1,5,2,3,4\}, \quad \{1,5,2,4,3\},\\ &\{1,5,3,2,4\}, \quad \{1,5,3,4,2\}, \quad \{1,5,4,2,3\}, \quad \{1,5,4,3,2\}\}\end{aligned}$$

First, let us examine the energy terms present in Q_j^{kin}. Since any complex microstate c chosen for a complex type j will always have the same number of total strands $L_j =$

$\sum_{i \in \Psi^0} A_{ij}$ (which is equivalent to $L(c)$), we should be able to extract the ΔG_{volume} term out of the exponential. Here we use the reference $\Delta G_{volume} = RT \log M_s$, where M_s is the number of solvent molecules in the fixed volume (see Section 3.1). Expanding $\Delta G^*(c)$ in equation 5.2, we get:

$$\begin{aligned} \exp(-\Delta G^*(c)/RT) &= \exp(-\overline{\Delta G}(c)/RT + (1 - L(c)) * \log M_s) \\ &= \frac{M_s}{M_s^{L(c)}} * \exp(-\overline{\Delta G}(c)/RT) \end{aligned} \tag{5.3}$$

Now, our summation for Q_j^{kin} is over complex microstates and thus where we consider all strands to be unique. In order to make the summation over the states $\Omega_j(\pi)$, we must see how many complex microstates there are that represent each state in $\Omega_j(\pi)$. A complex microstate c which has a indistinguishable ordering $\pi(c)$ (the circular ordering on the strand types) will correspond to exactly a single state $c' \in \Omega_j(\pi(c))$. So the question is how many such complex microstates c'' correspond to the same c'?

For a complex microstate c'' to correspond to the same c', we know that $\pi(c'') = \pi(c)$ and that there must exist a one-to-one mapping ξ between the strand ids of c and the strand ids of c'' such that for every base pair $(i_j \cdot k_l)$ in c, there is a base pair $(i_{\xi(j)} \cdot k_{\xi(l)})$ in c'', and for every base pair $(i_{\xi(j)} \cdot k_{\xi(l)})$ in c'' there is a base pair $(i_j \cdot k_l)$ in c. Finally this mapping must only map strand ids onto strand ids which share the same strand type. In other words, the mapping on strand ids induces a one-to-one mapping between the base pairs of each complex such that strands are always mapped to the same type of strands.

How many such mappings ξ are there for a given complex microstate c? If the state c' has no rotational symmetry, any permutation on the strand ids that maps strands to strands of the same type must be valid. Thus if c' has no rotational symmetry, there are exactly $\left(\prod_{i \in \Psi^0} A_{ij}!\right)$ complex microstates c'' which correspond to a given $c' \in \Omega_j(\pi(c))$. What about when c' has rotational symmetry $R(c')$? If this is the case, we know that there are $R(c')$ cyclic permutations on the indistinguishable ordering $\pi(c')$ which lead to the exact same structure (e.g., the same base pairings when we consider the strands to be indistinguishable). Thus if there is this symmetry present, we know that there must be $1/R(c')$ times as many complex microstates c'' which correspond to c' than if there were no rotational symmetry.

Thus, we can rewrite Q_j^{kin} using the above factors and equations 5.1 and 5.3:

$$\begin{aligned}
Q_j^{kin} &= \sum_{\pi \in \overline{\Pi}_j} \sum_{c \in \overline{\Omega}_j(\pi)} \exp(-\Delta G^*(c)/RT) \\
&= \sum_{\pi \in \Pi_j} \sum_{c \in \Omega_j(\pi)} \left(\prod_{i \in \Psi^0} A_{ij}! \right) * \frac{1}{R(c)} * \exp(-\Delta G^*(c)/RT) \\
&= \left(\prod_{i \in \Psi^0} A_{ij}! \right) \sum_{\pi \in \Pi_j} \sum_{c \in \Omega_j(\pi)} \frac{M_s}{M_s^{L_j}} * \frac{1}{R(c)} * \exp(-\overline{\Delta G}(c)/RT) \\
&= \frac{M_s}{M_s^{L_j}} * \left(\prod_{i \in \Psi^0} A_{ij}! \right) \sum_{\pi \in \Pi_j} \sum_{c \in \Omega_j(\pi)} * \exp(-\overline{\Delta G}(c)/RT - \log R(c)) \\
&= \frac{M_s}{M_s^{L_j}} * \left(\prod_{i \in \Psi^0} A_{ij}! \right) \sum_{\pi \in \Pi_j} \sum_{c \in \Omega_j(\pi)} * \exp(-(\overline{\Delta G}(c) - RT \log R(c))/RT) \\
&= \frac{M_s}{M_s^{L_j}} * \left(\prod_{i \in \Psi^0} A_{ij}! \right) \sum_{\pi \in \Pi_j} \sum_{c \in \Omega_j(\pi)} * \exp(-\Delta G(c)/RT) \\
&= \frac{M_s}{M_s^{L_j}} * \left(\prod_{i \in \Psi^0} A_{ij}! \right) Q_j \qquad (5.4)
\end{aligned}$$

And we have now shown that Q_j^{kin} is directly related to Q_j with a scaling factor of $\frac{M_s}{M_s^{L_j}}$ due to complex microstates having a ΔG_{volume} term in their energy, and a scaling factor of $\left(\prod_{i \in \Psi^0} A_{ij}!\right)$ due to the number of complex microstates which correspond to the same complex state when we consider strands of the same type to be indistinguishable.

5.3.2 Composing Q_{kin} from Q_j^{kin}

We begin by breaking down Q_{kin} into pieces based on the population vectors $m \in \Lambda$:

$$Q_{kin} = \sum_{m \in \Lambda} q_{kin}(m) \qquad (5.5)$$

where $q_{kin}(m)$ is the partition function over all system microstates s which have the population vector m. We now wish to show how to break those up in terms of the partition functions Q_j^{kin} which are for an arbitrary set of unique identifiers for a complex type j. So if our population vector contains multiple complex types that use the same strand type i,

we then must determine the number of ways to distribute the respective identifiers with that strand type to the complex types. And once we know which strand ids are being used for each strand type in a complex type j, we would have to distribute those among each complex of that type j which was present in the population vector.

For each strand type i, the number of ways to distribute the unique ids for that strand type to the multiset of complex types j in a population vector m is $\frac{m_i^0!}{\prod_{j\in\Psi}(A_{ij}*m_j)!}$. This is just the number of ways to distribute the m_i^0 distinct unique ids for that strand type into many piles, which have sizes $\{A_{ij}*m_j \| j \in \Psi\}$. We note that $A_{ij}*m_j$ is A_{ij}, the number of strand ids of type i in each complex of type j, multiplied by the m_j complexes of that type present in our population vector.

Thus over all strand types, the number of ways to distribute the unique ids among strands in the system to all complex types j in a population vector m is:

$$\prod_{i\in\Psi^0} \frac{m_i^0!}{\prod_{j\in\Psi}(A_{ij}*m_j)!} \tag{5.6}$$

Now that we have a fixed set of strand ids for a given complex type j, we need to determine the number of ways to distribute those strand ids to the m_j complexes which have that type.

For each strand type i the number of ways to distribute the $A_{ij}*m_j$ unique ids we have been given for our complex type j among the m_j complexes is $\frac{(A_{ij}*m_j)!}{(A_{ij}!)^{m_j}}$, if we have considered each complex to be uniquely labeled. Note that each complex is actually identical (and thus not uniquely labeled) when we have yet to assign any strand ids to it; however once we have assigned some (non-zero) strands of type i to the m_j different complexes, they will then be uniquely labeled based on which unique labels for strand type i have been assigned. Thus the first time we assign strand ids to a complex type j we must have $\frac{1}{m_j!}$ fewer ways of doing the assignment. We note that $\frac{(A_{ij}*m_j)!}{(A_{ij}!)^{m_j}}$ is the number of ways to distribute $A_{ij}*m_j$ objects among m_j uniquely labeled containers of size A_{ij} each.

Thus over all strand types, the number of ways to distribute the unique ids assigned to a complex type j to the m_j complexes which have that type is:

$$\frac{1}{m_j!} * \prod_{i\in\Psi^0} \frac{(A_{ij}*m_j)!}{(A_{ij}!)^{m_j}} \tag{5.7}$$

Finally, using these two equations (5.6 and 5.7) to count how many ways we could assign the strand ids among the different complex types in a population vector, we can finally write out $q_{kin}(m)$. $q_{kin}(m)$ is broken down into three terms: The number of ways we can distribute the strand ids to each complex type j (equation 5.6), multiplied by the product over, for each complex type j in Ψ, the partition function for that complex type j, $\left(Q_j^{kin}\right)^{m_j}$, multiplied by the number of ways we can assign the strand ids for that complex type to the m_j copies of that type (equation 5.7). This leads us to the following:

$$
\begin{aligned}
q_{kin}(m) &= \left(\prod_{i\in\Psi^0} \frac{m_i^0!}{\prod_{j\in\Psi}(A_{ij}*m_j)!}\right) * \left(\prod_{j\in\Psi}\left(Q_j^{kin}\right)^{m_j} * \frac{1}{m_j!} * \prod_{i\in\Psi^0}\frac{(A_{ij}*m_j)!}{(A_{ij}!)^{m_j}}\right) \\
&= \left(\prod_{i\in\Psi^0} m_i^0!\right) * \left(\prod_{j\in\Psi}\left(Q_j^{kin}\right)^{m_j} * \frac{1}{m_j!} * \prod_{i\in\Psi^0}\left(\frac{1}{(A_{ij}*m_j)!}\right) * \frac{(A_{ij}*m_j)!}{(A_{ij}!)^{m_j}}\right) \\
&= \left(\prod_{i\in\Psi^0} m_i^0!\right) * \left(\prod_{j\in\Psi}\left(Q_j^{kin}\right)^{m_j} * \frac{1}{m_j!} * \prod_{i\in\Psi^0}\frac{1}{(A_{ij}!)^{m_j}}\right) \\
&= \left(\prod_{i\in\Psi^0} m_i^0!\right) * \left(\prod_{j\in\Psi}\left(Q_j * \frac{M_s}{M_s^{L_j}} * \left(\prod_{i\in\Psi^0} A_{ij}!\right)\right)^{m_j} * \frac{1}{m_j!} * \prod_{i\in\Psi^0}\frac{1}{(A_{ij}!)^{m_j}}\right) \\
&= \left(\prod_{i\in\Psi^0} m_i^0!\right) * \left(\prod_{j\in\Psi} Q_j^{m_j} * \frac{M_s^{m_j}}{M_s^{L_j*m_j}} * \left(\prod_{i\in\Psi^0}(A_{ij}!)^{m_j}\right) * \frac{1}{m_j!} * \prod_{i\in\Psi^0}\frac{1}{(A_{ij}!)^{m_j}}\right) \\
&= \left(\prod_{i\in\Psi^0} m_i^0!\right) * \left(\prod_{j\in\Psi} Q_j^{m_j} * \frac{M_s^{m_j}}{M_s^{L_j*m_j}} * \frac{1}{m_j!}\right) \\
&= \left(\prod_{i\in\Psi^0} \frac{m_i^0!}{M_j^{m_i^0}}\right) * \left(\prod_{j\in\Psi} \frac{M_s^{m_j}}{m_j!} * Q_j^{m_j}\right) \qquad (5.8)
\end{aligned}
$$

For the final step above, note that $\sum_{i\in\Psi^0} m_i^0$ is the total number of strands present in the system as a whole, and $L_j * m_j$ is the total number of strands in all complexes of type j, so $\sum_{j\in\Psi} L_j * m_j$ must also be the total number of strands present in the system, thus $\prod_{i\in\Psi^0} \frac{1}{M_j^{m_i^0}} = \prod_{j\in\Psi} \frac{1}{M_s^{L_j*m_j}}$.

5.4 Conclusion

We now wish to compare against the thermodynamic Q_{box} from Dirks, et al. [6], which, when we assume $M_s \gg \sum_{i \in \Psi^0} m_i^0$ (which is also assumed in our model when we let $\Delta G_{volume} = RT \log M_s$), is:

$$Q_{box} = Q_{ref} * \sum_{m \in \Lambda} q(m)$$

where

$$q(m) \equiv \prod_{j \in \Psi} \frac{M_s^{m_j}}{m_j!} * Q_j^{m_j}$$

For the standard reference state where the ΔG_{box} is 0 when all strands are contained in the box and there are no base pairs, we have:

$$Q_{ref} \equiv \prod_{i \in \Psi^0} \frac{m_i^0!}{M_j^{m_i^0}}$$

Thus we can rewrite our Q_{kin} from equation 5.5 and equation 5.8, to get:

$$\begin{aligned}
Q_{kin} &= \sum_{m \in \Lambda} q_{kin}(m) \\
&= \sum_{m \in \Lambda} \left(\prod_{i \in \Psi^0} \frac{m_i^0!}{M_j^{m_i^0}} \right) * \left(\prod_{j \in \Psi} \frac{M_s^{m_j}}{m_j!} * Q_j^{m_j} \right) \\
&= \sum_{m \in \Lambda} Q_{ref} * q(m) \\
&= Q_{ref} * \sum_{m \in \Lambda} q(m) \\
&= Q_{box}
\end{aligned}$$

Thus we conclude that our partition function Q_{kin} over all system microstates is exactly equivalent to the thermodynamic partition function of the box, Q_{box}.

Additionally, we know that in the thermodynamic system (Dirks, et al. [6], equation 3.4), the probability of observing a population vector m at equilibrium is:

$$p(m) = Q_{box}^{-1} * Q_{ref} * q(m)$$

In any Markov process that satisfies detailed balance given enough time we will reach the thermodynamic equilibrium at which the probability of observing a system microstate s will obey the Boltzmann distribution $p_{kin}(s) = e^{-\Delta G_{box}(s)/RT}/Q_{kin}$, where $\Delta G_{box}(s) = \sum_{c \in s} \Delta G^*(c)$. (In reference [6], we note that ΔG_{box} appears with no arguments and has a different meaning, in which case $\Delta G_{box} = -RT \log Q_{box}$). So if we wish to find the probability of observing a particular population vector m, $p_{kin}(m)$, we take the sum over the probability of every system microstate s consistent with m (let us call this set $S(m)$). Thus we have $p_{kin}(m) = (1/Q_{kin}) * \sum_{s \in S(m)} e^{-\Delta G_{box}(s)/RT}$. The summation is exactly $q_{kin}(m)$ and so we have:

$$\begin{aligned} p_{kin}(m) &= \frac{q_{kin}(m)}{Q_{kin}} \\ &= \frac{Q_{ref} * q(m)}{Q_{kin}} \\ &= \frac{Q_{ref} * q(m)}{Q_{box}} \\ &= p(m) \end{aligned}$$

And thus we also compute the correct probability for observing a population vector m when using our model.

Chapter 6

The Simulator: Multistrand

Energy and kinetics models similar to these can been solved analytically; however, the standard master equation methods [30] scale with the size of the system's state space. For our DNA secondary structure state space, the size gets exponentially large as the strand length increases, so these methods become computationally prohibitive. One alternate method we can use is stochastic simulation [9], which has previously been done for single-stranded DNA and RNA folding (the **Kinfold** simulator [8]). Our stochastic simulation refines these methods for our particular energetics and kinetics models, which extends the simulator to handle systems with multiple strands and takes advantage of the localized energy model for DNA and RNA.

6.1 Data Structures

There are two main pieces that go into this new stochastic simulator. The first piece is the multiple data structures needed for the simulation: the *loop graph* which represents the complex microstates contained within a system microstate (Section 6.1.2), the *moves* which represent transitions in our kinetics model – the single base pair changes in our structure that are the basic step in the Markov process, and the *move tree* the container for moves that lets us efficiently store and organize them (Section 6.1.3).

6.1.1 Energy Model

Since the basic step for calculating the rate of a move involves the computation of a state's energy, we must be able to handle the energy model parameter set in a manner that simplifies this computation. Previous kinetic simulations (Kinfold) rely on the energy model we have

described, though without the extension to multiple strand systems. While the format of the parameter set that is used remains the same, we must implement an interface to this data which allows us to quickly compute the energy for particular loop structures (local components of the secondary structure, described in 3.2). This allows us to do the energy computations needed to compute the kinetic rates for individual components of the system microstate, allowing us to use more efficient algorithms for recomputing the energy and moves available to a state after each Markov step.

The energy model parameter set and calculations are implemented in a simple modular data structure that allows for both the energy computations at a local scale as we have previously mentioned, but also as a flexible subunit that can be extended to handle energy model parameter sets from different sources. In particular, we have implemented two particular parameter set sources: the NUPACK parameter set [34] and the Vienna RNA parameters [10] (which does not include multistranded parameters, so defaults for those are used). Adding new parameter set sources (such as the mfold parameters [37]) is a simple extension of the existing source code. Additionally, the energy model interface allows for easy extension of existing models to handle new situations, e.g., adding a sequence dependent term for long hairpins. We hope this energy model interface will be useful for future research where authors may wish to simulate systems with a unique energy model and kinetics model.

6.1.2 The Current State: Loop Structure

A system microstate can be stored in many different ways, as shown in Figure 6.1. Each of these has different advantages: the flat ("dot-paren") representation (Figure 6.1C) can be used for both the input and output of non-pseudoknotted structures, but the information contained in the representation needs additional processing to be used in an energy computation (we must break it into loops). Base pair list representation (Figure 6.1B) allows the definition of secondary structures which include pseudoknots, but also requires processing for energy computation. Loop representation (Figure 6.1D) allows the energy to be computed and stored in local components, but requires processing to obtain the global structure, used in input and output. While the loop graph cannot represent pseudoknotted structures without introducing a loop type for pseudoknots (for which we may not know how to calculate the energy), and making the loop graph cyclic, since this work is primar-

ily concerned with non-pseudoknotted structures this is only a minor point. In the future when we have excellent pseudoknot energy models, we will have to revisit this choice and hopefully find a good representation that still allows us similar computational efficiency.

We use the loop graph representation for each complex within a system microstate, and organize those with a simple list. This gives us the advantage that the energy can be computed for each individual node in the graph, and since each move only affects a small portion of the graph (Figure 6.3), we will only have to compute the energy for the affected nodes. While providing useful output of the current state then requires processing of the graph, it turns out to be a constant time operation if we store a flat representation which

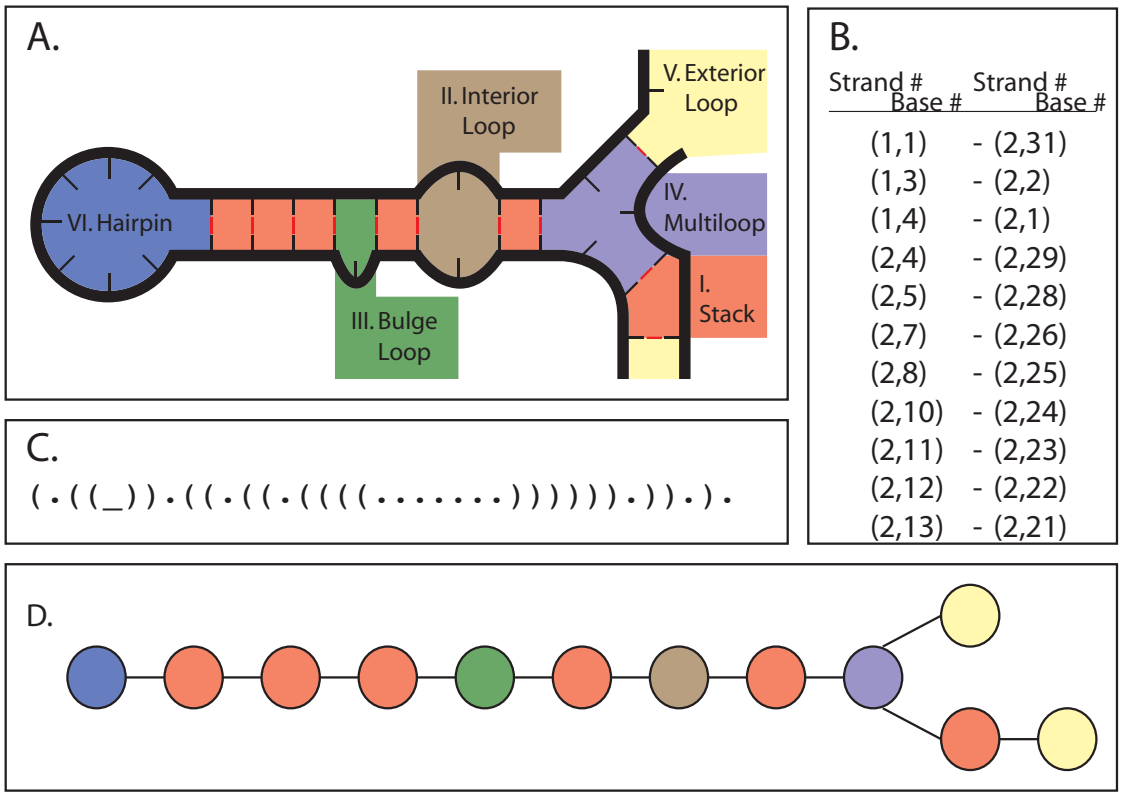


Figure 6.1: Example secondary structure, with different representations: (A) Original loop diagram representation. (B) Base pair list representation. Each base pairing is represented by the indices of the bases involved. (C) Dot-paren representation, also called the flat representation. Each base is represented by either a period, representing an unpaired base, or by a parenthesis, representing a pairing with the base that has the (balanced) matching parenthesis. An underscore represents a break between multiple strands. (D) Loop graph representation. Each loop in the secondary structure is a single node in the graph, which contains the sequence information within the loop.

gets updated incrementally as each move is performed by the simulator.

We contrast this approach with that in the original Kinfold, which uses a flat representation augmented by the base pairing list computed from it. Since we use a loop graph augmented by a flat representation, our space requirements are clearly greater, but only in a linear fashion: for each base pair in the list, we have exactly two loop nodes which must include the same information and the sequence data in that region.

6.1.3 Reachable States: Moves

When dealing with a flat representation or base pair list for a current state, we can simply store an available move as the indices of the bases involved in the move, as well as the rate at which the transition should occur. This approach is very straightforward to implement (as was done in the original Kinfold), and we can store all of the moves for the current state in a single global structure such as a list. However, when our current state is represented as a loop graph this simple representation can work, but does not contain enough information to efficiently identify the loops affected by the move. Thus we elect to add enough complexity to how we store the moves so that we can quickly identify the affected nodes in our loop graph, which allows us to quickly identify the loops for which we need to recalculate the available moves.

We let each move contain a reference to the loop(s) it affects (Figure 6.2A), as well as an index to the bases within the loop, such that we can uniquely identify the structural change that should be performed if this move is chosen. This reference allows us to quickly find the affected loop(s) once a move is chosen. We then collect all the moves which affect a particular loop and store them in a container associated with the loop (Figure 6.2B). This allows us to quickly access all the moves associated with a loop whose structure is being modified by the current move. We should note that since deletion moves by nature affect the two loops adjacent to the base pair being deletion, they must necessarily show up in the available moves for either loop. This is handled by including a copy of the deletion move in each loop's moves, and halving the rate at which each occurs.

Finally, since this method of move storage is not a global structure, we add a final layer of complexity on top, so that we can easily access all the moves available from the current state without needing to traverse the loop graph. This is as simple as storing each loop's move container in a larger structure such as a list or a tree, which represents the entire

complex's available moves as shown in Figure 6.2C.

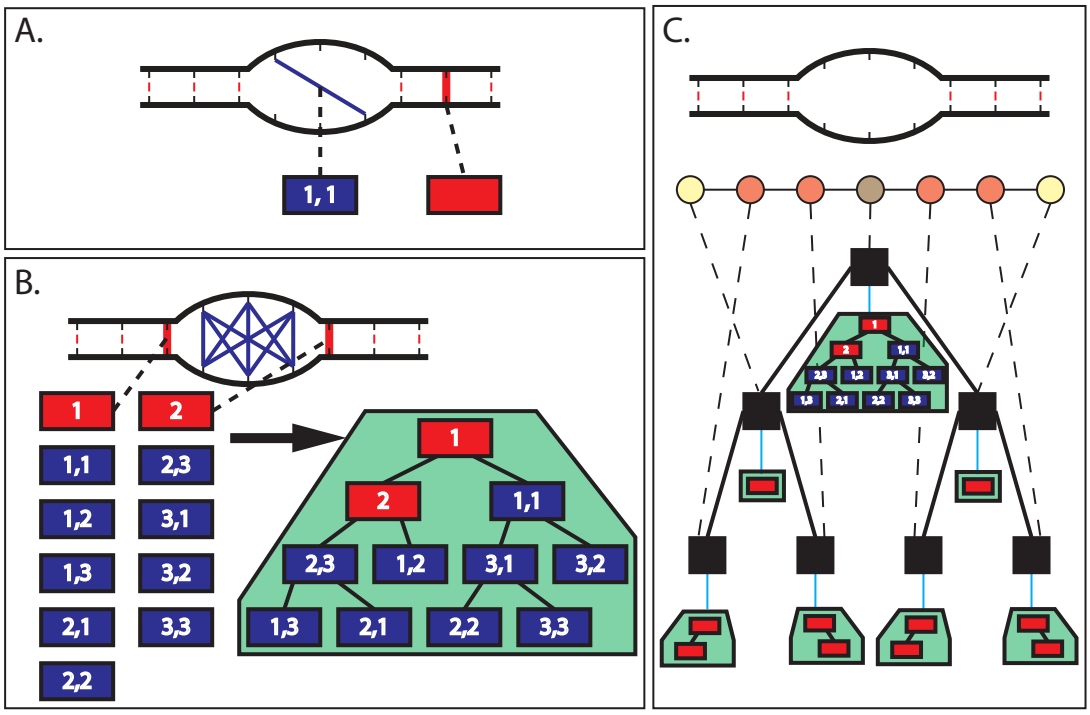


Figure 6.2: (A) Creation moves (blue line) and deletion moves (red highlight) are represented here by rectangles. Either type of move is associated with a particular loop, and has indices to designate which bases within the loop are affected. (B) All possible moves which affect the interior loop in the center of the structure. These are then arranged into a tree (green area), which can be used to quickly choose a move. (C) Each loop in the loop graph then has a tree of moves that affect it, and we can arrange these into another tree (black boxes), each node of which is associated with a particular loop (dashed line) and thus a tree of moves (blue line). This resulting tree then contains all the moves available in the complex.

6.2 Algorithms

The second main piece of the simulator is the algorithms that control the individual steps of the simulator. The algorithm implementing the Markov process simulation closely follows the Gillespie algorithm[9] in structure:

1. Initialization: Generate the initial loop graph representing the input state, and compute the possible transitions.

2. Stochastic Step: Generate random numbers to determine the next transition (6.2.1), as well as the time interval elapsed before the transition occurs.

3. Update: Change the current loop graph to reflect the chosen move (6.2.2). Recompute the available transitions from the new state (6.2.3). Update the current time using the time interval in the previous step.

4. Check Stopping Conditions: check if we are at some predetermined stopping condition (such as a maximum amount of simulated time) and stop if it is met. Otherwise, go back to step 2. These stopping conditions and other considerations relating to providing output are discussed further in Section 7.

The striking difference between this structure and the Gillespie algorithm is the necessity of recomputing the possible transitions from the current state at every step, and the complexity of that recalculation. Since we are dealing with an exponential state space we have no hope of storing all possible transitions between any possible pair of states, and instead must look at the transitions that occur only around the current state. Our examination of the key algorithms must include an analysis of their efficiency, so we define the key terms here:

1. N, the total length of the input's sequence(s).

2. X, the total amount of simulation time for each Monte Carlo trajectory.

3. J, the number of nodes in the loop graph of the current state. At worst case, this is $O(N)$, which occurs in highly structured configurations, like a full duplex.

4. K, the largest number of unpaired bases occurring in any loop in the current state. At worst case, this is exactly N, but on average it is typically much smaller.

5. C, the current number of complexes in the system. At worst this could be $O(N)$, but in practice the number of complexes is much fewer.

6.2.1 Move Selection

First let's look at the unimolecular moves in the system. The tree-based data structure containing the unimolecular moves leads to a simple choice algorithm that uses the generated random number to make a decision at each level of the tree based on the relative rates of the moves in each branch. We have two levels of tree to perform the choice on, the first having J nodes (one for every loop graph node) which each hold the local move containers

for a particular loop, and the second having at most $O(K^2)$ nodes (the worst case number of moves possible within a single loop). Thus our selection algorithm for unimolecular moves takes $O(\log(J) + \log(K))$ time to arrive at a final decision.

What about the moves that take place between two different complexes? With our method of assigning rates for these moves, we know that regardless of the resulting structure, all possible moves of this type must occur at the same rate. Thus the main problem is knowing how many such moves exist and then efficiently selecting one.

How many such moves exist? This is a straightforward calculation: for each complex microstate in the system, we count the number of A, G, C and T bases present in open loops within the complex. For the sake of example, let's call these quantities c_A, c_G, c_C, c_T for a complex microstate c. Let's also define the total number of each base in the system as follows: $A_{total} = \sum_{c \in s} c_A$, etc., where s is the system microstate we are computing the moves for. We can now compute how many moves there are where (for example) an A base in complex c becomes paired with a T base in any other complex: $c_A * (T_{total} - c_T)$, that is, the number of A bases within c multiplied by the number of T bases present in all other complexes in the system. So the number of moves between c and any other complex in the system is then $c_A * (T_{total} - c_T) + c_G * (C_{total} - c_C) + c_C * (G_{total} - c_G) + c_T * (A_{total} - c_A)$ and if we allow GT pairs, there are two additional terms with the same form. While GT (or GU if using a RNA substrate) pairs are not allowed by default, there is an option that can be set to allow them. Summing over this quantity for each $c \in s$ we then get 2 times the total number of bimolecular moves (and in fact we can eliminate the redundancy by using the total open loop bases in complexes "after" c in our data structure, rather than the total open loop bases in all complexes other than c). Since we do this in an algorithmic manner, it is straightforward to uniquely identify any particular move we need by simply following this counting process in reverse.

What is the time complexity for this bimolecular move choice? It is straightforward to see that calculating the total bimolecular move rate is $O(C)$ (recall C is the number of complexes within the system). Slightly more complex is choosing the bimolecular move, which must also be $O(C)$, as it takes 2 traversals through the list of complexes to determine the pair of reacting complexes in the bimolecular step. We note that for typical C and bimolecular reaction rates (e.g., typical strand concentrations which set our ΔG_{volume}) this quantity is quite small relative to that for the unimolecular reactions.

Our move choice algorithm can now be summed up as follows: given our random choice, decide whether the next move is bimolecular (using the total number of such moves as a first step) or unimolecular (thus one of the ones stored in the tree). If it's bimolecular, reverse the counting process using the random number to pick the unique combination of open loops and bases involved in the bimolecular step. If it's a unimolecular step, pick a move out of the trees of moves for each complex in the system as discussed above.

6.2.2 Move Update

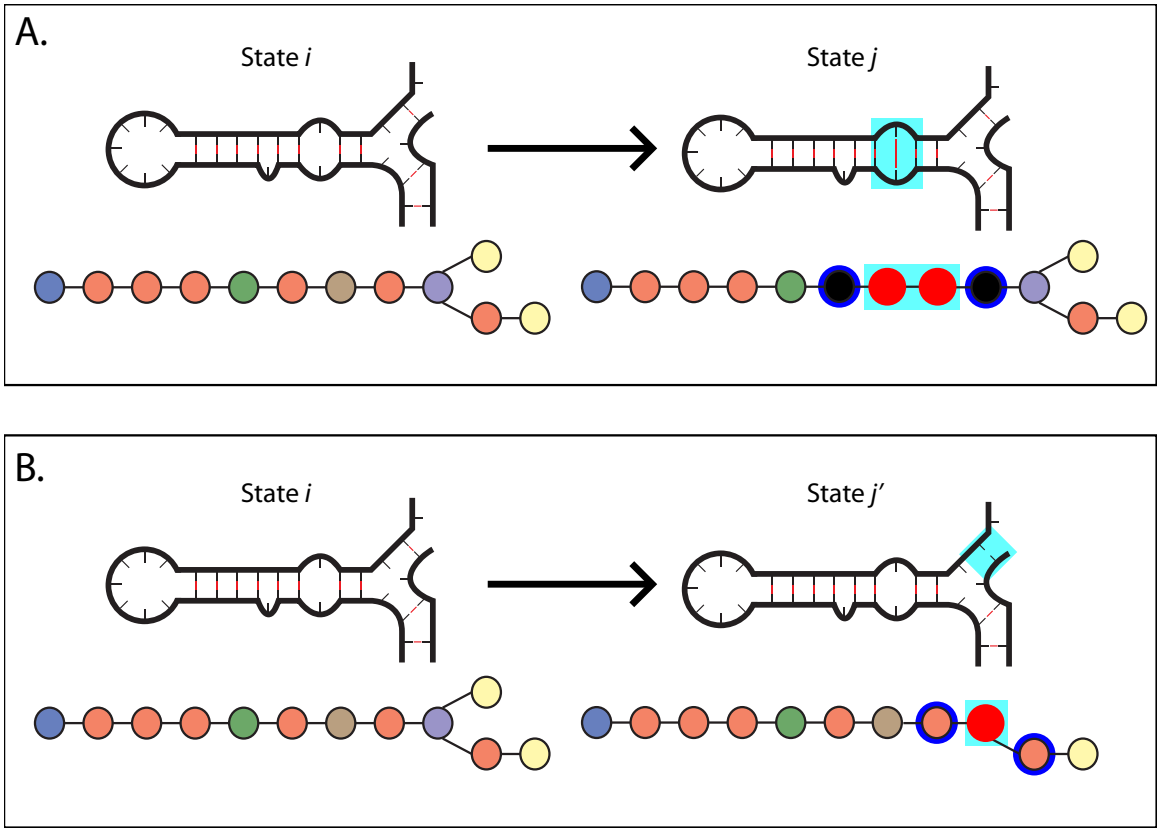


Figure 6.3: Moves of varying types which take current state i to state j. The changed region is highlighted in cyan. Loops that are in j but not i are highlighted in red (in the loop graph) and must be created and have their moves generated. Loops shown highlighted in blue have had an adjacent loop change, and thus must have their deletion moves recalculated. (A) A creation move. (B) A deletion move

Once a move has been chosen, we must update the loop graph to reflect the new state. This is a straightforward substitution: for a creation move, which affects a single loop, we must create the resulting pair of loops which replace the affected loop and update the graph connections appropriately (Figure 6.3A). Similarly, for a deletion move, which affects two

loops, we must create the single loop that results from joining the two affected loops, and update the graph connections appropriately (Figure 6.3B).

The computationally intensive part for this algorithm lies in the updating of the tree structure containing all the moves. We must remove the moves that involved the affected loops from the container, a process that takes $O(\log(J))$ time (assuming we implement tree deletions efficiently), generate the moves that correspond to the new loops (Section 6.2.3), and add these moves into the global move structure, which also takes $O(\log(J))$ time.

6.2.3 Move Generation

The creation and deletion moves must be generated for each new loop created by the move update algorithm, and we must update the deletion moves for each loop adjacent to an affected loop in the move update algorithm. The number of deletion moves which must be recalculated is fixed, though at worst case is linear in N, and so we will concern ourselves with the (typically) greater quantity of creation moves that need to be generated for the new loops.

For all types of loops, we can generate the creation moves by testing all possible combinations of two unpaired bases within the loop, tossing out those combinations which are too close together (and thus could not form even a hairpin, which requires at least three bases within the hairpin), and those for which the bases could not actually pair (for example, a T–T pairing). An example of this is shown for a simple interior loop, in Figure 6.4. The remaining combinations are all valid, and we must compute the energy of the loops which would result if the base pair were formed, in order to compute the rate using one of the kinetic rate methods (Section 4.5). This means we need to check $O(C^2)$ possible moves and do two loop energy computations for each. At worst case, that is $O(N^2)$ energy computations in this step, and so the efficiency of performing an energy computation becomes vitally important.

Once we have generated these moves we must collect them into a tree which represents the new loop's available moves. This can be handled in a linear fashion in the number of moves with a simple tree creation algorithm, and thus it is in the same order as the number of energy computations.

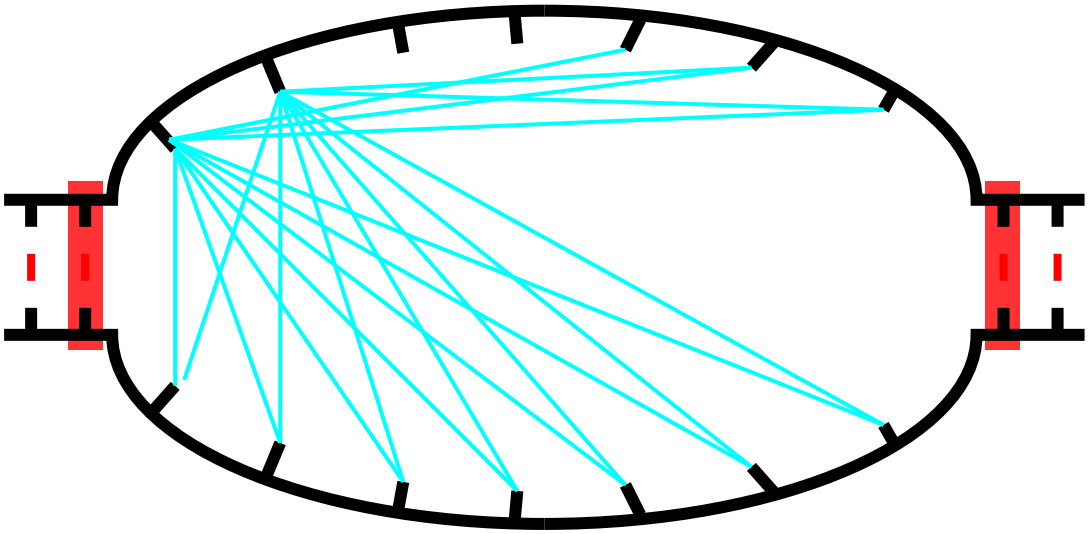


Figure 6.4: A interior loop, with all theoretically possible creation moves for the first two bases on the top strand shown as cyan lines, and all possible deletion moves shown as red boxes. Note that for each creation move shown here, we must check whether the bases could actually pair, and only then continue with the energy computations to determine the rate of the move.

6.2.4 Energy Computation

It is in the energy computation that our loop graph representation of the complex microstate shines, as the basic operation required for each possible move in the move generation step is computing the difference of energies between the affected loop(s) that would be present after the move and those present beforehand.

For all loop types except open loops and multiloops, computing the loop energy is as simple as looking up the sequence information (base pairs and adjacent bases) and loop sizes in a table [20, 37], and is a constant-time lookup. For open loops and multiloops, this computation is linear in the number of adjacent helices (e.g., adjacent nodes in the loop graph) if we are using an optional configuration of the energy model which adds energies that come from bases adjacent to the base pairs within the loop (called the "dangles" option). Theoretically we could have an open loop or multiloop that is adjacent to $O(N)$ other nodes in the loop graph, but this is an extraordinarily unlikely situation and present only with particular energy model options, so we will consider the energy computation step to be $O(1)$.

6.3 Time Complexity

Now that we have examined each algorithm needed to perform a single step of the stochastic simulation, we can derive the worst-case time. First, recall that J is the number of nodes in the loop graph, K is the largest number of unpaired bases in a loop, C is the number of complexes in the system and N is the total length of strands in the system. The move selection algorithm is $O(\log(J) + \log(K) + C) = O(N)$, move update is $O(\log(J)) = O(\log(N))$, and move generation is $O(N^2 * O(1))$, where energy computation is the $O(1)$ term. These algorithms are done in sequence and thus their times are additive: $O(N) + O(\log(N)) + O(N^2) = O(N^2)$. Thus our worst case time for a **single** Markov step is quadratic in the number of bases in our structure.

However, one step does not a kinetic trajectory make. We are attempting to simulate for a fixed amount of time X, as mentioned before, and so we must compute the expected number of steps needed to reach this time. Since the distribution on the time interval Δt between steps is an exponential distribution with rate parameter λ, which is the total rate of all moves in the current state, we know that the expected $\Delta t = 1/\lambda$. However, this still leaves us needing to approximate λ in order to compute the needed amount of time for an entire trajectory. To make a worst case estimate, we must use the largest λ that occurs in any given trajectory, as this provides the lower bound on the mean of the smallest time step Δt in that trajectory. However, the relative rates of favorable moves tends to be highly dependent on the rate method used: the Kawasaki method can have very large rates for a single move, while the Metropolis method has a maximum rate for any move, and the entropy/enthalpy method is also bounded in this manner as all moves have to overcome an energy barrier.

We thus make an average case estimate for the total rate λ, based on the number of "favorable" moves that typically have the largest rates. While a "favorable" move is merely one where the ΔG is negative (thus it results in an energetically more favorable state) or one which uses the maximum rate (for non-Kawasaki methods), the actual rate for these moves depends on the model chosen. The key question is whether we can come up with an average situation where there are $O(N^2)$ favorable moves or if $O(N)$ is more likely. What types of secondary structures give rise to quadratic numbers of moves? They are all situations where there are long unpaired sequences, whether in an interior loop, multiloop

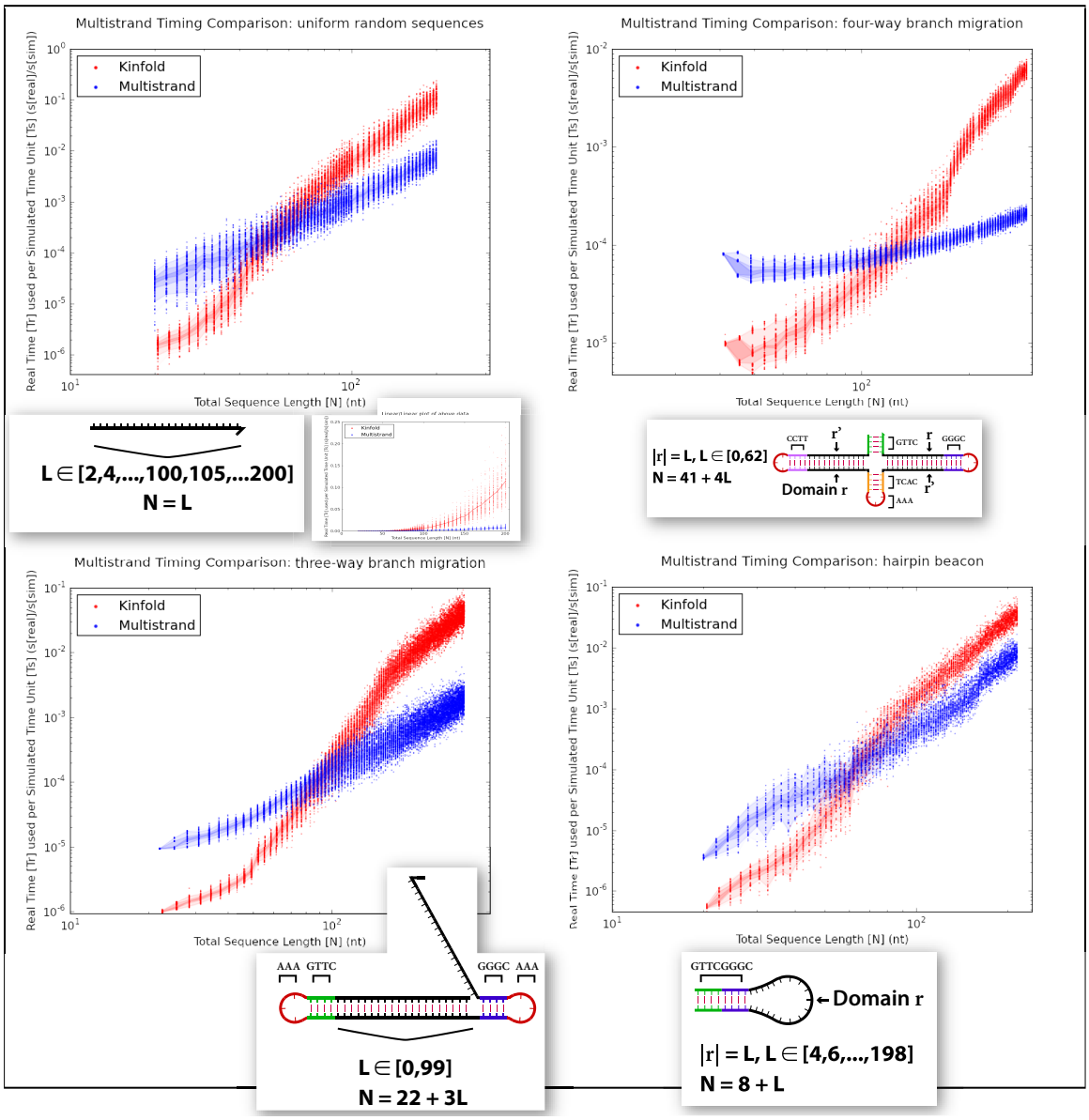


Figure 6.5: Comparison of real time used per simulated time unit between Multistrand and Kinfold 1.0, for four different single stranded systems with varying total length. All plots are log/log except for the inset, which is a linear/linear plot of the same data is in the uniform random sequence plot. Each individual point is a single randomly chosen sequence with the given length, and there are at most 100 such random sequences for each length. The density of these random sequences is shown using overlaid regions of varying intensity. From lightest to darkest, these correspond to 80%, 40%, and 10% of the random sequences being within the region.

or open loop. These creation moves are generally unfavorable, except for the small number that lead to new stack loops. Thus we do not expect there to be a quadratic number of favorable moves. A linear number is much more likely: a long duplex region could reach a

linear number (if say, $\frac{N}{4}$ bases were unpaired but could be formed easily into stacks). Thus, we make the (weak) argument that a good average case is that the average rate is at worst $O(N)$.

From this estimate for the average rate, we conclude that each step would have an expected (lower bound) $\Delta t = 1/N$, and thus to simulate for time X we would need $X/(1/N) = X * N$ steps, and thus $O(X * N^3)$ time to simulate a single trajectory. Since this is the worst case behavior, it is fairly difficult to show this with actual simulation data, so instead we present a comprehensive comparison with the original Kinfold for a variety of test systems (Figure 6.5), noting that the resulting slopes on the log/log plots lie easily within the $O(N^3)$ bound for the time taken to simulate 1 time unit. In this plot we can also distinguish the behavior on systems where we are close to the worst-case estimates (when most of the bases are in large single stranded regions), as we expect for the "uniform random sequences" case, and those where we are far from the worst-case (most bases are in highly structured regions), such as in the "four-way branch migration" case.

Chapter 7

Multistrand: Output and Analysis

We have now presented the models and algorithms that form the continuous time Markov process simulator. Now we move on to discuss the most important part of the simulator from a user's perspective: the huge volume of data produced by the simulation, and methods for processing that data into useful information for analyzing the simulated system.

How much data are we talking about here? Following the discussion in the previous chapter, we expect an average of $O(N)$ moves per time unit simulated. This doesn't tell us much about the actual amount of data, only that we expect it to not change drastically for different size input systems. In practice this amount can be quite large, even for simple systems: for a simple 25 base hairpin sequence (similar to Figure 6.5D), it takes 4,000,000 Markov steps to simulate 1s of real time. For an even larger system, such as a four-way branch migration system (Figure 6.5C) with 108 total bases, simulating 1s of real time takes 14,000,000 Markov steps.

What can we do with all the data produced by the simulator? In the following sections we discuss several different processing methods.

7.1 Trajectory Mode

This full trajectory information can be useful to the user in several ways: finding kinetic traps in the system, visualizing a kinetic pathway, or as raw data to be passed to another analysis tool.

Trajectory mode is Multistrand's simplest output mode. The data produced by this mode is a trajectory through the secondary structure state space. While many trajectories could be produced for a given system, for most analysis purposes discussed in this section

we are only concerned with a single trajectory. Similarly, these trajectories are potentially infinite but unfortunately our computers have only a finite amount of storage so we must cut the trajectory off at some point.

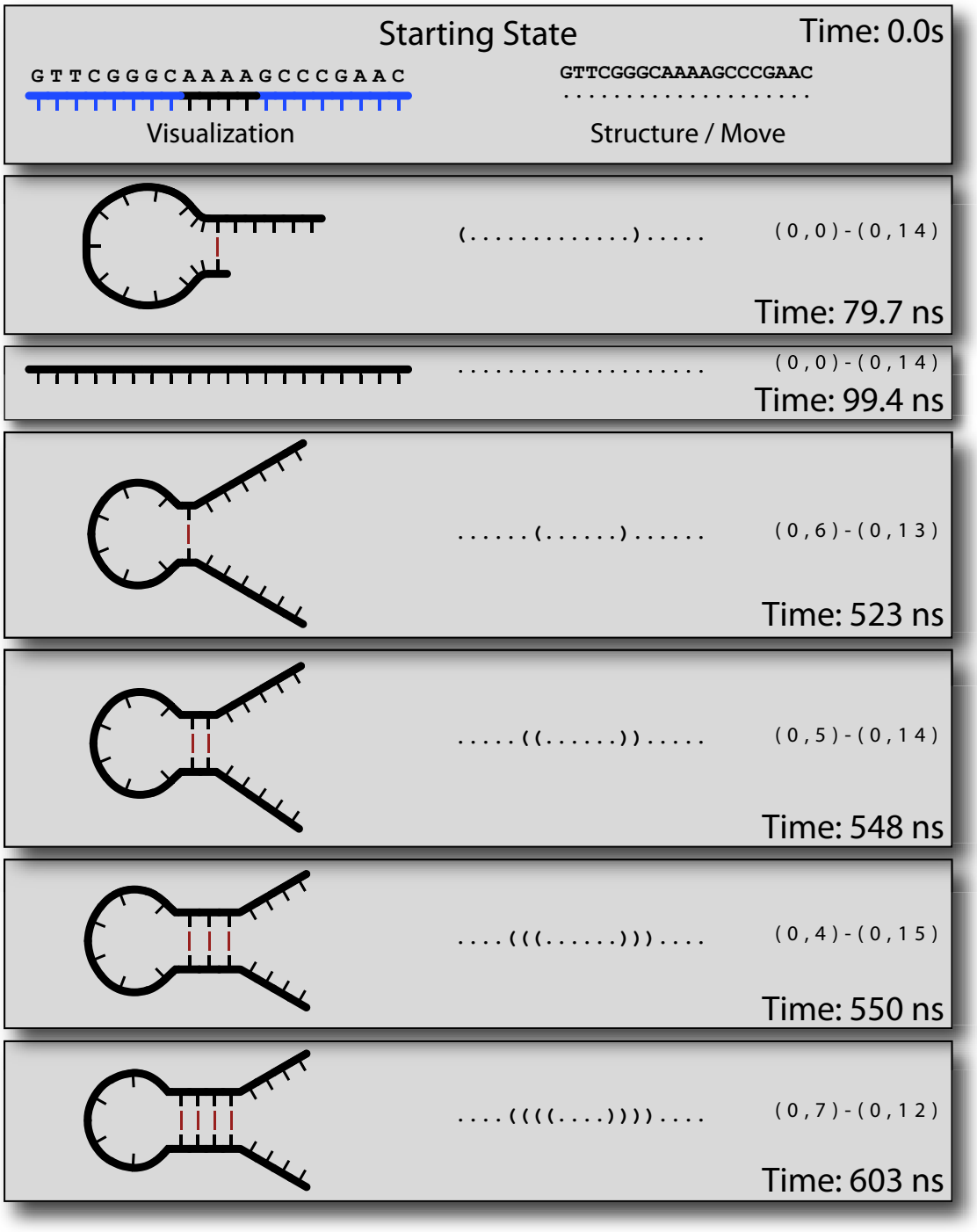


Figure 7.1: Trajectory data

A trajectory is represented by a finite ordered list of (s, t) pairs, where s is a system

microstate, and t is the time in the simulation at which that state is reached. We call this time the *simulation time*, as opposed to the *wall clock time*, the real world time it has taken to simulate the trajectory up to that point. There are many different ways to represent a trajectory, as shown in Figure 7.1.

For practical reasons, we set up conditions to stop the simulation so that our trajectories are finite. There are two basic stop conditions that can be used, and the system stops when any condition is met:

1. Maximum simulation time. We set a maximum simulation time t' for a trajectory, and stop when the current simulation state (s, t) has $t > t'$. Note that the state (s, t) which caused the stopping condition to be met is not included in the trajectory, as it is different from the state at time t'.

2. Stop state. Given a system microstate s', we stop the trajectory when the current simulation state (s, t) has $s = s'$. This type of stopping condition can be specified multiple times, with a new system microstate s' each time; the simulation will stop when any of the provided microstates is reached.

We will now use an example to show how trajectory mode can be used to compare two different sequence designs for a particular system. The system is a straightforward three-way branch migration with three strands, with a six base toehold region and twenty base branch migration region, shown below (Figure 7.2).

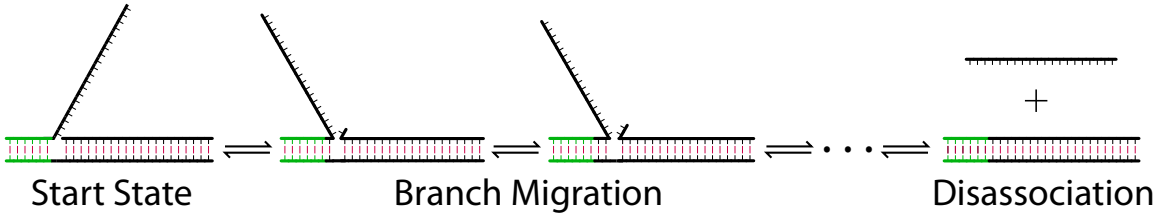


Figure 7.2: Three-way branch migration system. The toehold region is in green, and the branch migration region is black. A few intermediate states along a sample trajectory are shown, with transition arrows indicating not a single base-pair step but a pair of steps that break one base-pair then form another. Many possible side reactions also exist, such as breathing of duplex regions and sequence-dependent hairpin formation within the single-stranded region.

The simulation is started in the shown **Start State** using a toehold sequence of `GTGGGT` and a differing branch migration region for which we use the designs in Table 7.1. We then start trajectory mode for each design, with a stop condition of 0.05 s of simulation time, and save the resulting trajectories.

	Branch Migration Region
Design A	`ACCGCACGTCCACGGTGTCG`
Design B	`ACCGCACCACGTGGGTGTCG`

Table 7.1: Two different branch migration sequences

Rather than spam the interested reader with several thousand pages of trajectory printouts, since there are $5*10^6$ states in a 0.05 s trajectory for this system, we instead highlight one revealing section in each design's trajectory. Let us look at the state the trajectory is in after 0.01 s of simulation time, shown below in Figure 7.3 using a visual representation.

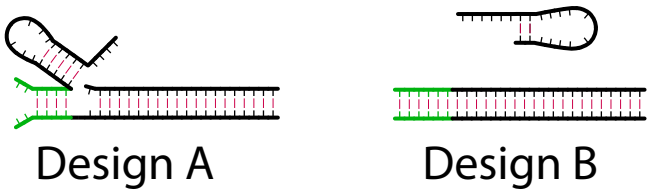


Figure 7.3: Structure after 0.01 s simulation time for two different sequence designs

What happened? It appears that sequence design A has a structure that can form before the branch migration process initiates, that contains a hairpin in the single stranded branch migration region. Does this structure prevent the branch migration from completing? In the long run it shouldn't, as the equilibrium structure remains unchanged, but if we look at the final state in each trajectory (Figure 7.4), we see that design B has completed the process in 0.05 s of simulation time and indeed was complete at 0.01 s, where A is still stuck in that offending structure after the same amount of time. So for these specific trajectories, it's certainly slowing down the branch migration process.

Did this structure only appear because we were unlucky in the trajectory for design A? We could try running several more trajectories and seeing whether it appears in all or most of them, but a more complete answer is better handled using a different simulation mode, such as the first passage time mode discussed in Section 7.4.

A better type of question for trajectory mode is "How did this kinetic trap form?" In this example, we can examine the trajectory for design A and find the sequence of system

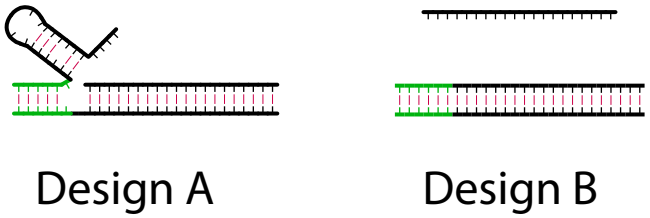


Figure 7.4: Final structure (0.05 s simulation time) for the two different sequence designs from Table 7.1. Branch migration regions: Design A: ACCGCACGTCCACGGTGTCG, Design B: ACCGCACCACGTGGGTGTCG

microstates that lead to the first time the hairpin structure forms. This example has a straightforward answer: the competing structure forms before the branch migration starts, and is therefore in direct competition with the correct kinetic pathway.

We expect that the most common usage for trajectory mode is in providing the raw trajectory data for a separate tool to perform processing on. For example, taking the raw trajectory data and producing a movie of the structure's conformational changes can be very helpful in visualizing a system, and also is quite helpful for examining kinetic traps. A quick movie of the three-way branch migration system could identify how the kinetic trap forms, rather than our examination of thousands of states by hand to locate that point.

7.1.1 Testing: Energy Model

We have also used the trajectory mode to aid in verifying that the kinetics model and energy model was implemented correctly. For the energy model, we can use an augmented output that includes the Multistrand-calculated energy for a given state, and compare that to the energy predicted by NUPACK [34] (or whichever tool / source we are using for our energy parameter dataset). This can be done using trajectory mode, with a cutoff time of 0 s, so the initial state is the only one in each trajectory. Multistrand's energy model was verified to be consistent with NUPACK for every sequence and structure in a comprehensive test set of secondary structures (part of the NUPACK package) that covers all possible loop configurations.

7.1.2 Testing: Kinetics Model

Testing the kinetics model can be done by testing that the *detailed balance* condition in fact holds: We know that at equilibrium, if our kinetics model obeys detailed balance,

the distribution of states seen by our simulator (after sufficient time to reach equilibrium) should agree with the Boltzmann distribution on each system microstate's energy. There are several ways we could extract this information from trajectory mode, such as recording all microstates seen in the trajectory (perhaps after some minimum time) and the amount of time spent in each one.

For our testing of the detailed balance condition we use a different method that is simpler to implement: we run many trajectories with a fixed maximum simulation time t and record only the final state in the trajectory (note that this is the state at time t in the trajectory, **not** the state which caused the stopping condition to be met). Assuming that the time t is large enough for us to reach equilibrium, we can compare the probability distribution over the final states seen by the simulation to that predicted using the NUPACK partition function and energy calculation utilities. In particular, for each final state observed in a trajectory we count the number of times it occurred as a final state in our simulation, and use that to compute the simulation probability for that state. We then calculate the thermodynamic probability of observing that state using the NUPACK tools. Finally, we take the absolute value of the difference between the thermodynamic probability and the simulation probability for each final state observed and sum those quantities to obtain the total probability difference between our simulator and the thermodynamic predictions.

For our single-stranded test cases we found this probability difference to be less than 1% when running a sufficient number of trajectories (approximately 10^5). This measure steadily decreases with increased trajectory count, and does not change when the simulation time is exponentially increased, indicating that our chosen t was enough to reach an equilibrium state and the probability difference is due to the stochastic nature of the simulation. The states which we observed accounted for 99.95% of the partition function, and that percentage also increases with increased number of trajectories.

7.2 Macrostates

In Section 2.3 we defined a *system microstate*, which represents the configuration (primary and secondary structure) of the strands in the simulation volume. In this section, we will define a *macrostate* of the system and show how these objects can help us analyze a system by providing better stop states, as well as allowing new avenues of analysis, as discussed

in Section 7.3. To make things simpler in this section, when we refer to a *microstate* we always mean a system microstate unless stated otherwise.

Formally, we define a *macrostate* m as a non-empty set of microstates: $m = \{s_1, s_2, ..., s_n\}$, where each s_i is a microstate of the system. Now we wish to derive the free energy of a macrostate, $\Delta G(m)$ in such a way that the probability of observing the macrostate m at equilibrium is consistent with probability of observing any of the contained microstates.

$$\begin{aligned} Pr(m) = & \quad Pr(s_1) + Pr(s_2) + \ldots + Pr(s_n) \\ = & \quad \sum_{1 \leq i \leq n} Pr(s_i) \\ = & \quad \sum_{1 \leq i \leq n} \frac{1}{Q_{kin}} * e^{-\Delta G_{box}(s_i)/RT} \\ = & \quad \frac{1}{Q_{kin}} * \sum_{1 \leq i \leq n} e^{-\Delta G_{box}(s_i)/RT} \end{aligned} \tag{7.1}$$

Now, letting $Q_m = \sum_{1 \leq i \leq n} e^{-\Delta G_{box}(s_i)/RT}$, the partition function of the macrostate m, we have $Pr(m) = \frac{Q_m}{Q_{kin}}$. Similarly, in terms of the energy of the macrostate, we can express $Pr(m)$ as $\frac{1}{Q_{kin}} * e^{-\Delta G(m)/RT}$, and plugging into (7.1) and solving for $\Delta G(m)$, we get:

$$\begin{aligned} \frac{1}{Q_{kin}} * e^{-\Delta G(m)/RT} = & \quad \frac{1}{Q_{kin}} * Q_m \\ e^{-\Delta G(m)/RT} = & \quad Q_m \\ -\Delta G(m)/RT = & \quad \log Q_m \\ \Delta G(m) = & \quad -RT * \log Q_m \end{aligned} \tag{7.2}$$

Now that we have the formal definition in place, let us look at an example macrostate using the same three-way branch migration system as in the previous section, Figure 7.2.

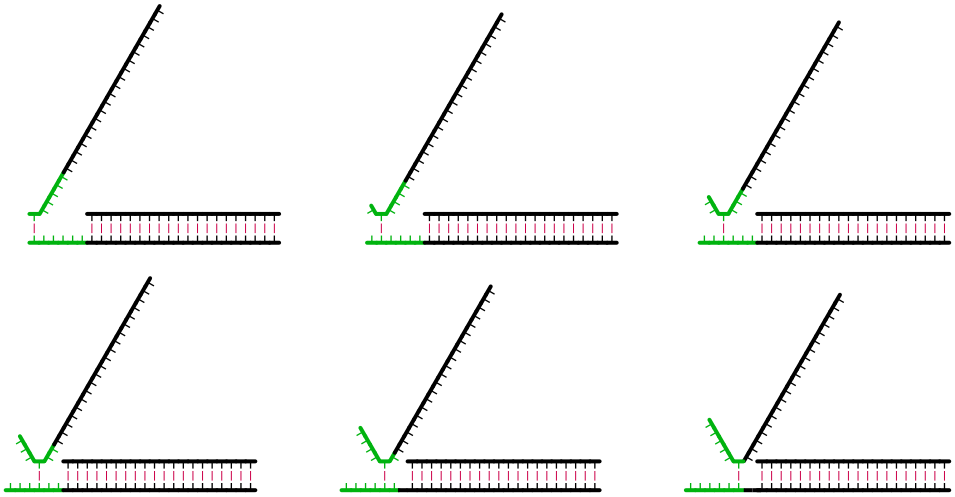
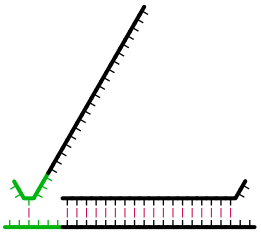


Figure 7.5: Example macrostate

What does this macrostate represent? It's a set of microstates that has exactly one base pair formed in the toehold region, but it's not every such microstate – every microstate shown has the entire branch migration region fully formed. Thus the following microstate isn't included in the macrostate, but it does have exactly one base pair formed in the toehold region:



Why are these general macrostates interesting? Previously, we defined stop states as being microstates of the system, and we can use any number of them as part of the simulator's stop conditions. From that, it's easy to see that any given macrostate m could be used as a stop state of the system by simply expanding it out into the list of microstates contained within and using those as individual stop states.

Of particular interest to us are several classes of macrostates which can be described in very simple terms and also checked efficiently by the simulator *without* having to individually check for each microstate within those macrostates. The ability to check for a macrostate efficiently is very important: if we allowed the branch migration region in the previous example to have any structure, the macrostate would contain over 2^{22} microstates, and

even if we allowed only a limited number of bases in the branch migration region to be breathing (such as 3 base pairs, e.g., 6 bases) this is still 1140 microstates.

One useful tool in defining these classes of macrostates is a distance metric for comparing two complex microstates c_i, c_j. The distance $d(c_i, c_j)$ is defined as ∞ if c_i and c_j do not have the same set of strands and strand ordering, and otherwise as the number of bases which are paired differently between the two structures: e.g., if base x is paired with base y in c_i, but base x isn't paired with y in c_j, or if base x is unpaired in c_i, but base x is paired in c_j. This distance metric has been used in other work, using a slightly different but equivalent formulation, for example, [6, 13] and references therein. Some examples are shown below, in Table 7.2.

c_i	Structure	Distance
c_0	....((((_))))....	
c_1	(((((((((_)))))))))	$d(c_0, c_1) = 8$
c_2	....((((_)).....))	$d(c_0, c_2) = 6$
c_3	.......((_)).....	$d(c_0, c_3) = 4$
c_4	....((((_)))).....)	$d(c_0, c_4) = 3$
c_5	.(....)(_)........	$d(c_0, c_5) = 7$

Table 7.2: Distance metric examples, for complex microstates on the two-strand complex with sequences AGCTAGCT,AGCTAGCT. Bases that differ from the structure c_0 are shown in red.

Now that we have a distance metric, we define several common macrostates that can be used in the simulator as stopping conditions.

7.2.1 Common Macrostates

Disassoc: Given a set of strands ST and an ordering π^* on those strands, we define the **Disassoc** macrostate m as the set of all system microstates s which contain a complex microstate c with exactly the strands ST and ordering π^*. Recall that a complex microstate (Section 2.2) is defined by three quantities, the strands contained in the connected complex, the ordering on those strands, and the base pairs present; thus this definition implies no particular set of base pairs are present, though it does require that the complex be connected. Note that this macrostate can only be reached by either an association or a disassociation step, allowing it to be efficiently checked as we only need to do so when encountering a bimolecular move. It's called **Disassoc** in

light of its most common usage, but it could also be used to stop after an association event.

Bound: Given a single strand S, we define the **Bound** macrostate m as the set of all system microstates s which contain a complex microstate c with set of strands ST that has $S \in ST$ and $|ST| > 1$.

Count: Given a complex microstate c and an integer count k, we define the **Count** macrostate m as the set of all system microstates s which contain a complex microstate c' for which $d(c, c') \leq k$. Note that c' which meet this criteria must have the same strands and strand ordering, as $d(c, c') = \infty$ if they do not. For convenience, instead of using the integer count k we allow passing a percentage p which represents a percentage of the total number of bases N in the complex c. If this is done, we use a cutoff $k = \lceil p * N \rceil$.

Loose: Given a complex microstate c, a integer count k and a set of bases B that is a subset of all the bases in c, we define the **Loose** macrostate m as the set of all system microstates s which contain a complex microstate c' for which $d_B(c, c') \leq k$, where we define d_B as the distance metric d over only the set of bases B in c. Similar to the **Count** macrostate, we allow a percentage p instead of k, for which we set $k = \lceil p * |B| \rceil$. This macrostate allows us to specify a specific region of interest in a microstate, such as just a toehold region we wish to be bound without caring about other areas in the complex microstate.

Note that each of these macrostates is based on the properties of a single complex microstate occurring within a system microstate; thus if desired we could make a stopping condition which uses several of these in conjunction. For example, we might make a stopping conditions that has **Disassoc** for strand A and **Disassoc** for strand B, thus creating a macrostate which can be described in words as "strand A is in a complex by itself, and strand B is in a complex by itself, and we don't care about any other parts of the system". Similarly we can implement disjunction simply by using multiple independent stopping conditions. Though the **NOT** operation is not currently implemented for these stop conditions, it may be added in the future, allowing us to have the full range of Boolean operations on these common macrostates. As it is, we can easily implement the original example macrostate

simply by using an **OR** of the six exact system microstates. Or we could use **Loose** macrostates to implement the one we might have intended, where we didn't care very much about the branch migration region (and thus allowed it to have some breathing base pairs), only that a single base of the toehold had been formed.

7.3 Transition Mode

What is transition mode? The basic idea is that instead of **every** system microstate being an interesting piece of the trajectory, we provide (as part of the input) a list T of *transition states* of the system, the states which we think are interesting, and the output is then the times when we enter or leave any transition state in the list T. These transition states can be exact states of the system (e.g., system microstates), or macrostates of the system (e.g., a combination of common macrostates such as **Dissasoc** or **Loose** macrostates), and we note that they are not required to be technical "transition states" as in chemical reaction theory—we are interested in how trajectories move (i.e., transition) from macrostate to macrostate, no matter how those macrostates are defined. One way to look at this form of output is as a trajectory across transition state membership vectors. We note that since these transition states are defined in exactly the same way as stop states, we generally lump them both together in the list of transition states that get reported (after all, you'd like to know what state caused the simulation to finish, right?), with a special labeling for which transition states are also stop states.

What is transition mode good for? The simplest answer is that it allows us to ask questions about specific kinetic pathways. Here's an example of this: Given a simple sequence that forms a hairpin, does it form starting from the bases closest to the 5'/3' ends (Figure 7.6B), or starting from the bases closest to the hairpin region (Figure 7.6C)?

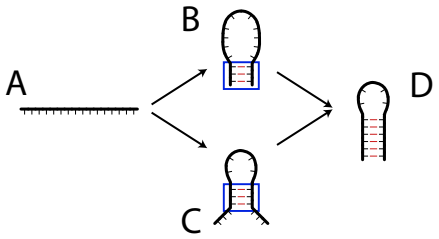


Figure 7.6: Hairpin Folding Pathways. Blue boxes indicate regions of interest used in loose structure definitions (Table 7.3). A) Starting State. B) Bases near the 5'/3' ends form first. C) Bases near the hairpin region form first. D) Final hairpin structure.

How do we represent these pathways in terms of transition states? Here we take advantage of the common macrostate definitions (Section 7.2.1) to define the intermediate structures B and C, using loose macrostates with a distance of 2, while A and D are defined with exact microstates.

Transition State Label	Sequence / Structure	State Type
	GCATGCAAAAGCATGC	
A (start)	.....	Exact
B	(((**********)))	Loose, $d \leq 2$
C	***(((****)))***	Loose, $d \leq 2$
D (stop)	((((((....))))))	Exact

Table 7.3: Transition states for hairpin pathway example. State type of **Exact** is exactly the given structure as a system microstate, and **Loose** is a loose macrostate covering only the bases in blue (or alternately, the bases not marked with "*").

Why is using a loose macrostate for these transition states useful? First, we note that we produce output any time the transition state membership changes, hence each step of the pathway is the set of all transition states which match the system microstate. Let's look at a possible pathway to the stop state where the first bases that form are near the 5' and 3' ends and the base pairs are added sequentially without ever being broken. With exact states this would result in the following transition pathway: $\{A\} \rightarrow \emptyset \rightarrow \{B\} \rightarrow \emptyset \rightarrow \{D\}$ and with the loose macrostates it would be this transition pathway: $\{A\} \rightarrow \emptyset \rightarrow \{B\} \rightarrow \{B, C\} \rightarrow \{B, C, D\}$. So far, so good. What about if we form two bases of B, then all of C, then the last base of B? For loose states, this is the exact same transition pathway—recall that we use a distance of 2, and two base pairs formed in B is exactly that distance away from the given structure. But for exact states, this is now the (very boring) pathway $\{A\} \rightarrow \emptyset \rightarrow \{D\}$, which doesn't answer our question about which part of the helix formed first!

Two possible transition pathways, using either the loose structures for B and C, or exact structures:

Time	Transition States
0.00	A
$3.63 * 10^{-7}$	$\emptyset$
$1.03 * 10^{-6}$	A
$1.40 * 10^{-6}$	$\emptyset$
$1.78 * 10^{-6}$	B
$1.92 * 10^{-6}$	B, C
$2.15 * 10^{-6}$	B, C, D

(a) Sample Transition Pathway (Loose)

Time	Transition States
0.00	A
$9.02 * 10^{-7}$	$\emptyset$
$1.31 * 10^{-6}$	A
$2.26 * 10^{-6}$	$\emptyset$
$2.72 * 10^{-6}$	D

(b) Sample Transition Pathway (Exact)

Table 7.4: Two different transition pathways via transition mode simulation, using either the given B and C states with the loose macrostate definitions from Table 7.3, or exact system microstates using the states from the same table with all "*" replaced by "." (unpaired) and distance set to 0, effectively. Note that the times listed are the times of first entering the given state.

Does this mean every simulated trajectory takes these transition pathways? Definitely not! The stochastic nature of the simulator means we're likely to see many different transition pathways if we run many trajectories. So, let's now answer the original question: which transition pathway is more likely? We do this by accumulating statistics over many kinetic trajectories as follows: For each transition path trajectory (such as those in Table 7.4) we break down the trajectory into pieces which have non-empty sets of transition states, separated only by zero or one empty set of transition states. So, for example, the path shown in Table 7.4a breaks down into four separate reactions: $\{A\} \to \emptyset \to \{A\}$, $\{A\} \to \emptyset \to \{B\}$, $\{B\} \to \{B, C\}$, and $\{B, C\} \to \{B, C, D\}$. For our statistics, we'll group reactions of the form $x \to \emptyset \to y$ with those of the form $x \to y$, and for every possible reaction, we record the number of times it occurred and the average time it took to occur. So for the single pathway in Table 7.4a we get the following statistics:

Reaction	Average Time	Number of Occurrences
$A \to A$	$1.03 * 10^{-6}$	1
$A \to B$	$7.43 * 10^{-7}$	1
$B \to B, C$	$1.47 * 10^{-7}$	1
$B, C \to B, C, D$	$2.29 * 10^{-7}$	1

Table 7.5: Statistics for the single transition pathway shown in Table 7.4a

Now that we've seen an example of these statistics for a single kinetic trajectory, let's look at the same statistics over a hundred kinetic trajectories, again using the system with loose macrostates.

Reaction	Average Time	Number of Occurrences
$A \to A$	$2.48 * 10^{-6}$	829
$A \to B$	$2.17 * 10^{-7}$	37
$A \to C$	$2.53 * 10^{-7}$	73
$B \to A$	$1.09 * 10^{-6}$	5
$B \to B$	$1.46 * 10^{-7}$	2
$B \to B, C$	$3.78 * 10^{-7}$	33
$C \to A$	$5.63 * 10^{-7}$	5
$C \to C$	$2.48 * 10^{-7}$	7
$C \to B, C$	$5.84 * 10^{-7}$	77
$B, C \to B$	$4.32 * 10^{-7}$	1
$B, C \to C$	$1.21 * 10^{-7}$	9
$B, C \to B, C, D$	$2.10 * 10^{-7}$	100

Table 7.6: Statistics for 100 simulated trajectories using the transition states from Table 7.3

What can we conclude from these statistics? Both pathways do occur, but it is much more likely that the first bases formed are those closest to the hairpin region. The average times for each pathway are roughly within an order of magnitude of each other, and our selection of transition states was good: we didn't see any unexpected pathways, such as $\{A\} \to \{D\}$.

We could use these "reactions" to create a coarse-grained representation of the original system as a chemical reaction network, using $\frac{1}{\text{avg time}}$ as the reaction rate constants. Whether this will be an accurate representation or not depends on the choice of transition states and the structure of the energy landscape. For example, if we were to try this using the average times for this system, we would end up with a formal CRN in which the $A \to A$ reaction is taken far less frequently than shown in Table 7.6. Finding appropriate coarse-grained representations is a deep and subtle topic [15].

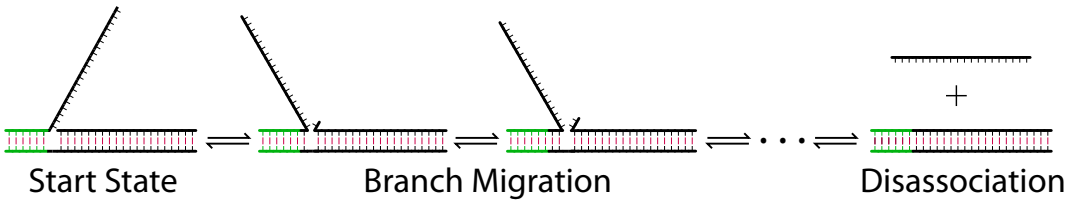
7.4 First Passage Time Mode

First passage time mode is the most basic simulation mode in Multistrand. It produces a single piece of data for each trajectory simulated: the first passage time for reaching any stop state in the system, and which stop state was reached. This is a rather striking difference from our previous simulation modes in the amount of data produced for each individual trajectory, but it is still quite powerful!

This first passage time data could be produced via trajectory mode: we can just discard

all the output until a stop state is reached. There is a distinct efficiency advantage to making it a separate simulation mode: we don't have to pay the overhead of reporting every piece of trajectory data only for it to be discarded. Similarly, we could generate the same data using transition mode by only using stop states in our list of transition states. We implement this as a distinct simulation mode in order to better separate the reasons for using each simulation mode: for transition mode, we are interested in the pathway our system takes to reach a stop state, and for first passage time mode we are interested in how quickly the system reaches the stop state(s).

What does first passage time data look like? Let's revisit our example system from Section 7.1 (Figure 7.2):



We start the system as shown, and use two different stop states: the **complete** stop condition where the incumbent strand has disassociated (as shown in the figure), and the **failed** stop condition where the invading strand has disassociated without completing the branch migration. Both of these are done using *Disassoc* macrostates, which makes it very efficient to check the stop states. Note that we include the invading strand disassociating as a stop state so that if it occurs (which should be very rarely), we can find out easily without waiting until the maximum simulation time or until the strands reassociate and complete the branch migration.

The following table (Table 7.7) shows a five trajectories worth of data from first passage time mode on the example system, using sequence design B (Table 7.1) for the branch migration region.

Note that we have included a third piece of data for each trajectory, which is the pseudorandom number generator seed used to simulate that trajectory. This allows us to produce the exact same trajectory again using a different simulation mode, stop states or other output conditions. For example, we might wish to run the fifth trajectory in the table again using trajectory mode, to see why it took longer than the others, or run the first

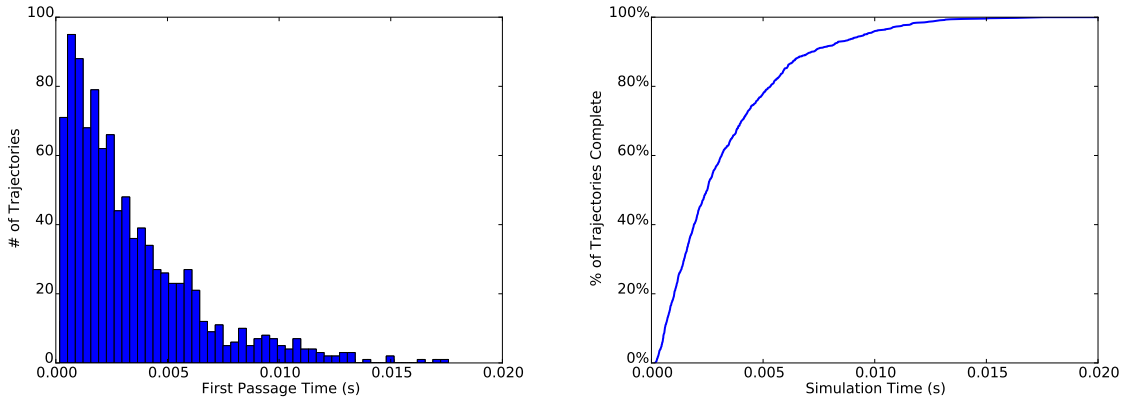
Random Number Seed	Completion Time	Stop Condition
`0x790e400d`	$3.7 * 10^{-3}$	failed
`0x38188213`	$3.8 * 10^{-3}$	complete
`0x47607ebf`	$2.1 * 10^{-3}$	complete
`0x02efe7fa`	$2.8 * 10^{-3}$	complete
`0x7c590233`	$6.7 * 10^{-3}$	complete

Table 7.7: First passage time data for the example three-way branch migration system. Stop conditions are either "complete", indicating the branch migration completed successfully, or "failed", indicating the strands fell apart before the branch migration could complete.

trajectory to see what kinetic pathway it took to reach the **failed** stop condition.

Let's now look at a much larger data set for first passage time mode. Here we again use the three-way branch migration system with sequence design B for the branch migration region and increase the toehold region to be ten bases, to minimize the number of trajectories that reach the **failed** stop condition. We run 1000 trajectories, using a maximum simulation time of 1 s, though no trajectory actually used that much as we shall shortly see.

Instead of listing all the trajectories in a table, we graph the first passage time data for the **complete** stop condition in two different ways: first (Figure 7.7a) we make a histogram of the distribution of first passage times for the data set, and second (Figure 7.7b) we graph the percentage of trajectories in our sample that have reached the **complete** stop condition as a function of the simulation time.



(a) Histogram of first passage times

(b) Percent completion by simulation time

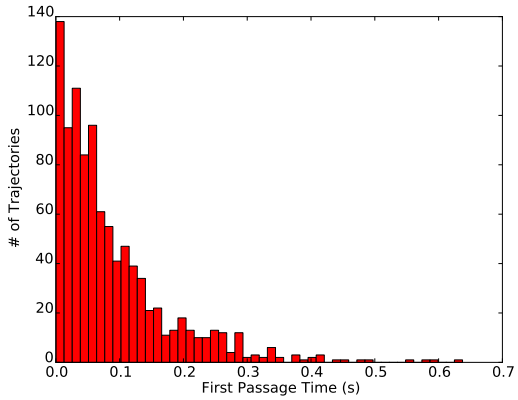
Figure 7.7: First passage time data for the three-way branch migration system, using sequence design B (Table 7.1) and with a ten base toehold sequence. 1000 trajectories were simulated and all of them ended with the **complete** stop condition.

While there are many ways to analyze these figures, we note two particular observations. Firstly, the histogram of the first passage time distribution looks suspiciously like an exponential distribution, possibly with a short delay. This is not always typical (as we shall shortly see), but the shape of this histogram can be very helpful in inferring how we might wish to model our system based on the simulation data; e.g., for this system, we might decide that the three-way branch migration process is roughly exponential (with some fitted rate parameter) and so we could model it as a one-step unimolecular process.

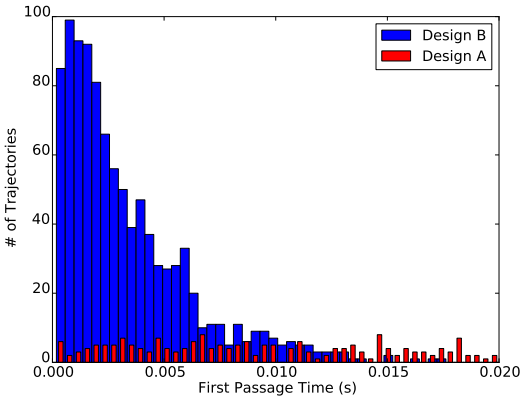
The second observation is that while the percentage completion graph looks very similar to an experimental fluorescence microscopy curve, they should **NOT** be assumed to be directly comparable. The main pitfall to watch out for is when comparing fluorescence curves from systems where the reactions are bimolecular: in these the concentration of the relevant molecules are changing over time, but in our stochastic simulation the bimolecular steps are at a fixed volume/concentration (reflected in the ΔG_{volume} energy term) and data is aggregated over many trajectories.

7.4.1 Comparing Sequence Designs

A common usage of first passage time mode is in the comparison of sequence designs, as we previously brought up in Section 7.1. We now run another 1000 trajectories on the same three-way branch migration system as in the previous section, including the increased toehold length, but using the sequence design A (Table 7.1) for the branch migration region. Note the change in x-axis scale; this design is indeed much slower than design B!



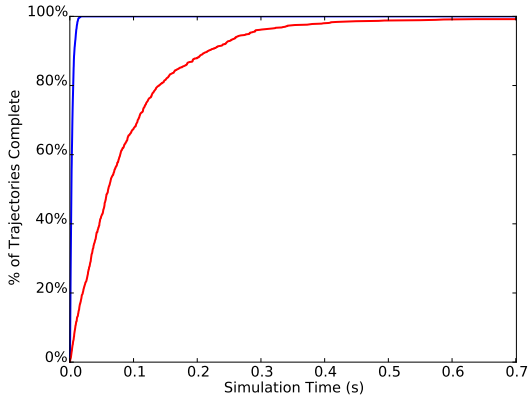
(a) Histogram of first passage times, design A



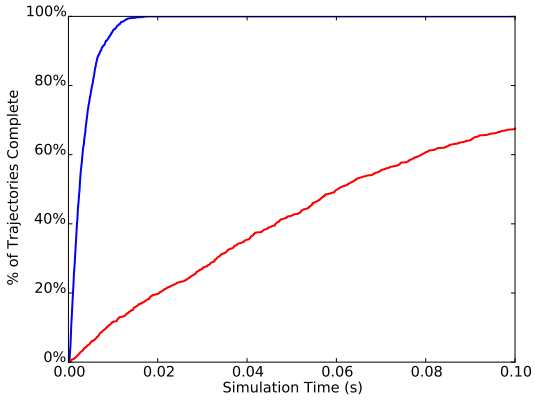
(b) Histogram of first passage times, both designs

Figure 7.8: First passage time data for the three-way branch migration system, comparing sequence designs using histograms. For figure (b), we compare the two designs on the range of times from 0 s to 0.02 s. The buckets for sequence design A have been reduced in visual size to show overlapping regions, but overall bucket sizes are consistent between the two designs (though they are slightly different from those in Figure 7.7a).

Let's also look at the same data but using the percentage completion as a function of simulated time graphs:



(a) Design comparison, percent completion graph



(b) Design comparison, zoomed

Figure 7.9: First passage time data for the three-way branch migration system, comparing sequence designs using percent completion graphs. Figure (b) is a zoom-in on the range of times from 0 s to 0.1 s from Figure (a).

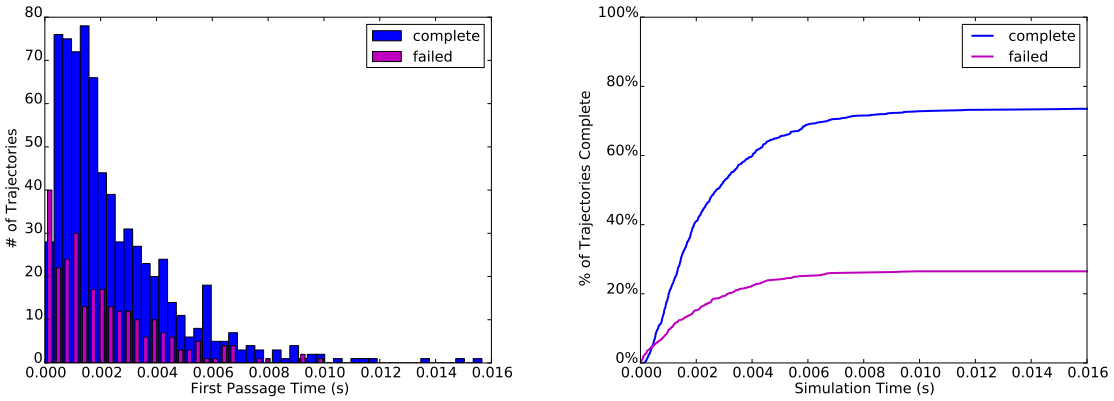
We can now clearly see just how different these two sequence designs are! Our results from the trajectory mode section were clearly not unusual: the majority of sequence design B's trajectories finish in under 0.01 s, whereas the same amount for sequence design A

is over ten times longer. This is highlighted in the combined graph (Figure 7.8b), which shows how vastly different the timescales are for each process. Looking at the percentage completion graphs, we note that sequence design A did not actually reach 100% – it actually had 8 trajectories reach the **failed** stop condition!

While both types of graphs are presenting the same information, they are frequently useful in different cases: the first passage time histograms are helpful for gaining an intuition into the actual distribution of times for the process we are simulating, while the percent completion graphs are better for looking at the relative rates of different designs, especially when working with more than two sequence designs.

7.4.2 Systems with Multiple Stop Conditions

What about our original three-way branch migration system, which had toeholds of length six? Let's now look at this situation, where we have a system that has more than one competing stop state. We again run the system with 1000 trajectories and a 1 s maximum simulation time (though it's never reached). Instead of plotting only the first passage times for the **complete** stop condition, we overlay those with the first passage times for the **failed** stop condition.



(a) Histogram of first passage times

(b) Percent completion by simulation time

Figure 7.10: First passage time data for the three-way branch migration system with 6 base toeholds, comparing sequence designs

Competing stop conditions certainly make the data more interesting! We can pick out pieces of the kinetic pathways a lot easier using these graphs. For example, the competing pathway leading to the **failed** stop condition frequently occurs faster than the **complete**

stop condition pathway; this should be unsurprising, as one pathway involves a long random walk while the other is very unlikely to include one.

7.5 Fitting Chemical Reaction Equations

We are frequently interested in DNA systems which can be represented (with some choice of the appropriate internal parameters/sequences) by chemical reaction equations of the following form:

$$A + B \xrightleftharpoons{k_{eff}} C + D \tag{7.3}$$

These systems usually involve an intermediate step, so typically the concentration is low enough for the above equation to actually be a good fit. Experimental observation of these systems tend to be in the range of concentrations where the above equation is an accurate characterization of the system.

Let us now associate the species in the equation above with actual DNA complexes (ordered collections of connected strands). We designate strands by unique letters, and indicate complementary strands with an asterisk. For the toehold-mediated three-way branch migration example, the chemical equation then becomes:

$$x^*y + x \xrightleftharpoons{k_{eff}} xx^* + y \tag{7.4}$$

Let us examine one possible DNA configuration for the state of the entire system that could follow this equation's implied dynamics. The left hand side could be the configuration given in Figure 7.11, and the right hand side could be the configuration given in Figure 7.12. Note that while we show particular exact secondary structures for each of these figures, there are actually many such secondary structures that represent the appropriate parts of the equation, which might be specified using **Loose** macrostates.

Finally, what would the equations look like if we did not expect them to fit equation 7.3? One possibility is as follows (as in Zhang, Winfree 2009 [35]):

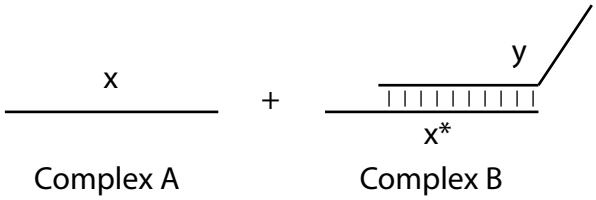


Figure 7.11: Starting complexes and strand labels

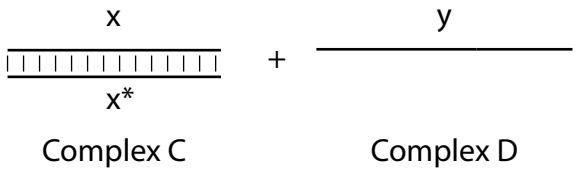


Figure 7.12: Final complexes and strand labels

$$A + B \underset{k_r}{\overset{k_f}{\rightleftharpoons}} AB \underset{k_{bm}}{\overset{k_{bm}}{\rightleftharpoons}} AB' \xrightarrow{k_r} C + D \tag{7.5}$$

Though this model is used in several experimental papers, it is difficult for us to define simulation macrostates in order to determine the various rates present in the equation. Instead, we look at a model where it is easy to separate out the steps into discrete components which can be individually simulated. Specifically, we look at a system where the molecules A and B can collide and form either a reactive molecule which will go to C and D, or a nonreactive molecule which will fall apart after some time[1]. We call this the **first step model**, and it is described by the equations below:

$$A + B \xrightarrow{k_1} AB \xrightarrow{k_2} C + D \tag{7.6}$$

$$A + B \underset{k_2'}{\overset{k_1'}{\rightleftharpoons}} AB' \tag{7.7}$$

We will use this model extensively to analyze the results of the **first step mode** simu-

[1]Thanks to Niles Pierce and Victor Beck for suggesting this approach.

lations discussed in Section 7.6. Note that for low concentrations, the controlling step will be the bimolecular reactions and thus we should be able to also fit to a k_{eff} type model in those cases.

We discuss fitting first passage time data to the simple k_{eff} model (Equation 7.3, as well as using **first step mode** to generate data which can easily be fit to the first step model.

7.5.1 Fitting Full Simulation Data to the k_{eff} Model

For this simulation mode, we start the simulation in the exact state shown in Figure 7.11, and measure the first passage time to reach any state with the same complexes (but not exact secondary structure, i.e., we use a **Disassoc** stop state) as that shown in Figure 7.12. This gives us a data set of first passage times Δt_i, where $1 \leq i \leq n$, and n is the total number of trajectories simulated. The simulation is done at a particular concentration z. Note that this simulation will require time inversely proportional to the simulation concentration, since we are simulating elementary steps. Thus we would prefer to simulate at higher concentrations and step downwards in concentration until we are in the region where the bimolecular reaction dominates, and equation 7.3 holds.

If we assume equation 7.3 holds and determines our distribution of first reaction times, we can fit our data to an exponential distribution in order to determine k_{eff}: Recall that (via Gillespie [9]) in a formal chemical reaction network, a state with total outgoing propensity a_0 will have a passage time (τ) distribution according to the probability density function $P(\tau) = a_0 * \exp(-a_0\tau)$. If we are given a set of first passage times Δt_i taken from this distribution, we know that for an exponential distribution that $a_0 = 1/\mathrm{E}[\Delta t_i]$. Since the propensity a_0 for a bimolecular reaction is the reaction rate times the concentration, we solve for k_{eff} as follows, where z is the simulated concentration:

$$k_{eff} = \frac{1}{\mathrm{E}[\Delta t_i]} * 1/z \tag{7.8}$$

If we are in the regime where equation 7.3 holds, we expect this to give us a consistent value for k_{eff}. If we are outside of that regime (and thus the unimolecular reactions from

equation 7.5 or equation 7.6 dominates), this will err by exactly the factor of $1/z$, thus the graph of k_{eff} versus concentration z should appear linear in this regime. For

7.6 First Step Mode

This simulation mode makes a simple assumption: we always start the Markov simulation by making a "join" step, that is, one where a pair of molecules A and B come together and form a single base pair. The choice of secondary structure states for the molecules A and B before collision can be done either by using particular exact complex microstates, or by Boltzmann sampling the secondary structure space of the molecules. This sampling is valid when the bimolecular reaction rates are slow enough that the initial complexes reach equilibrium. In either case, we have a valid system microstate once we know the structure for the A and B molecules, and then choose a "join" step from those present in the system microstate and use the resulting system microstate (after making the join) as our start state for a trajectory. Since this mode runs many trajectories, we must make many such random choices for the join step and for the Boltzmann sampling of the initial molecules (if we are using the sampling rather than exact states).

The simulation then starts from this configuration, and we track two distinct end states: the molecules falling apart back into one of the $A + B$ configurations, or the molecules reacting into one of the $C + D$ configurations. Our data then consists of first passage times where we can separate each trajectory into one that reacted or one that failed. The advantage to this mode of simulation is that we no longer are directly simulating the "join" bimolecular steps, whose rates are proportional to the simulated concentration and thus are going to be very slow relative to the normal unimolecular steps. This allows us to use the simulator to concentrate on the trajectories where we have a collision, rather than spending (at low concentrations) most of the time simulating unimolecular reactions while waiting for the very rare bimolecular reaction.

7.6.1 Fitting the First Step Model

Our first step mode simulation produces the following pieces of data: Δt^i_{react}, the first passage times for reactive trajectories, Δt^i_{fail} the first passage times for trajectories did not react, the number of trajectories that reacted N_{react} and failed N_{fail}, and the rate constant

k_{coll}, the simulation's estimate of the rate of collision (in $/M/s$) of the A and B molecules. This k_{coll} is calculated based on *all* the join moves possible from the initial configuration of the A and B molecules, and thus is very likely to include many such moves which do not lead to very stable structures and thus disassociate quickly.

We then fit to the model given in equations 7.6 and 7.7, as follows:

$$k_1 = \frac{N_{react}}{N_{react} + N_{fail}} * k_{coll} \tag{7.9}$$

$$k_1' = \frac{N_{fail}}{N_{react} + N_{fail}} * k_{coll} \tag{7.10}$$

$$k_2 = \frac{1}{\mathrm{E}[\Delta t^i_{react}]} \tag{7.11}$$

$$k_2' = \frac{1}{\mathrm{E}[\Delta t^i_{fail}]} \tag{7.12}$$

Thus we can directly find each of the model parameters from the collected simulation data, in a natural way. We then use this model to predict the k_{eff} parameter we would observe if we assume that the simple chemical reaction model (equation 7.3) is valid.

7.6.2 Analysis of First Step Model Parameters

We first show a natural (but inexact) way to calculate k_{eff} from the first step model parameters, using the assumption that the time used in "failed" collisions is negligible compared to that needed for a successful reaction. In this situation, we can estimate k_{eff} for a particular concentration z by calculating the expected average time for a reaction. We use the fact that the expected value of an exponential distribution with rate parameter λ will be $\frac{1}{\lambda}$ in order to derive k_{eff}:

$$k_{eff} = \frac{1}{z} * \frac{1}{\frac{1}{k_1 * z} + \frac{1}{k_2}} \tag{7.13}$$

Again, this makes the assumption that equation 7.3 holds and thus that the reaction dominated by the bimolecular step. The observation from the full simulation mode still holds: if we are not in this regime, we will err by a factor of $\frac{1}{z}$ and thus the graph should be linear. Note that since this simulation does not require a set concentration, we can run

one simulation (with a large number of trajectories) and use the extracted data to produce the same type of graph as the full simulation mode.

We now would like to remove the assumption that the "failed" collision time is negligible: though that assumption makes k_{eff} straightforward to calculate, many systems of interest will not satisfy that condition.

We now need to calculate the expected time for a "successful" reaction to occur based on both the "failed" and "reactive" collision parameters. We do this by summing over all possible paths through the reactions in equations 7.6 and 7.7, weighted by the probability of those reactions. Let $\Delta t_{coll} = \frac{1}{(k_1+k_1')*z}$ (the expected time for any collision to occur), $\Delta t_{fail} = \Delta t_{coll} + \frac{1}{k_2'}$ (the expected time needed for a failed collision to return to the initial state), $\Delta t_{react} = \Delta t_{coll} + \frac{1}{k_2}$ (the expected time for a reactive collision to reach the final state), $p(path)$ is the probability of a particular path occurring, and $\Delta t_{path} = n\Delta t_{fail} + \Delta t_{react}$ is the expected time for a path which has n failed collisions and then a successful collision. Finally, the quantity which we want to solve for is $\Delta t_{correct}$, the expected time it takes for a successful reaction to occur.

$$\Delta t_{correct} = \sum_{path} \Delta t_{path} * p(path) \tag{7.14}$$

$$= \sum_{n=0}^{\infty} (n\Delta t_{fail} + \Delta t_{react}) * (\frac{k_1'}{k_1 + k_1'})^n * \frac{k_1}{k_1 + k_1'} \tag{7.15}$$

To simplify the next step, let $\alpha = \frac{k_1}{k_1+k_1'}$ and $\alpha' = \frac{k_1'}{k_1+k_1'}$, and recall that for $\beta > 0$, $\sum_{n=0}^{\infty} \beta^n = \frac{1}{1-\beta}$ and $\sum_{n=0}^{\infty} n * \beta^n = \beta * \frac{1}{(1-\beta)^2}$, and we get:

$$\Delta t_{correct} = \Delta t_{fail} * \frac{\alpha'}{(1-\alpha')^2} * \alpha + \Delta t_{react} * \frac{1}{1-\alpha'} * \alpha \tag{7.16}$$

Now we note that $\frac{\alpha}{1-\alpha'} = 1$, and $\frac{\alpha'}{1-\alpha'} = \frac{k_1'}{k_1}$, and simplify:

$$\Delta t_{correct} = \Delta t_{fail} * \frac{k_1'}{k_1} + \Delta t_{react} \tag{7.17}$$

And thus we arrive at the (full) form for k_{eff}:

$$k_{eff} = \frac{1}{\Delta t_{correct}} * \frac{1}{z} \tag{7.18}$$

This requires only a single assumption: that the reaction is dominated by the bimolecular step, and thus can be described by equation 7.3. We note that this derivation can be generalized for multiple possible stop states [2].

Chapter 8

Calibration of Kinetic Parameters

8.1 k_{uni} Calibration

We calibrate the k_{uni} parameter using measurements of the zippering rate of DNA strands[29]. This rate is that of forming single base pairs at the end of a DNA helix, a process found very commonly during DNA hybridization where two complementary strands co-locate, form a nucleating base pair and then rapidly form base pairs (zippering) until the complementary strands are fully hybridized.

The reference we use for the zippering rate [29] provides a measurement of the rate at a temperature of 353 °K (approximately 80 °C), as well as the following equation (equation 40 in [29]) which we may use to obtain the zippering rate at other temperatures. We note that

$$k_{zip} = A_{zip} * \exp(E_{zip}/RT)$$

Since we are interested in the zippering rate at the standard reference temperature of 37 °C, we solve this equation for A_{zip} using their $k_{zip} = 3 * 10^9$ s^{-1} at 80 °C, and value of E_{zip} of 7.5 kcal/mol, and arrive at $A_{zip} = 1.329 * 10^{14}$ s^{-1}. Thus we have a zippering rate at 37 °C of $k_{zip} = 6.842 * 10^8$ s^{-1}. We will also be interested in simulations of room temperature reactions (25 °C), for which $k_{zip} = 4.191 * 10^8$ s^{-1}.

How do we then use this zippering rate to inform our choice of k_{uni}? Let us look at the rates predicted by our kinetic model for the zippering step, shown in Figure 8.1. Regardless of the rate method chosen, we know that our kinetics model for a step will predict a rate $k = k_{uni} * f(\Delta G_{step})$ where f is a function of the energy difference ΔG_{step} between the two

states and is a different function for different rate methods (see equations 4.12 and 4.15). For example, with Metropolis rules, $f(\Delta G_{step}) = 1$ when $\Delta G_{step} \leq 0.0$. In Wetmur and Davidson [29], the authors calculate their zippering rate by looking at all possible sequences with a weighting based on the base composition of the sequence. For the reference k_{zip} the GC content of the sequences was approximately 50%.

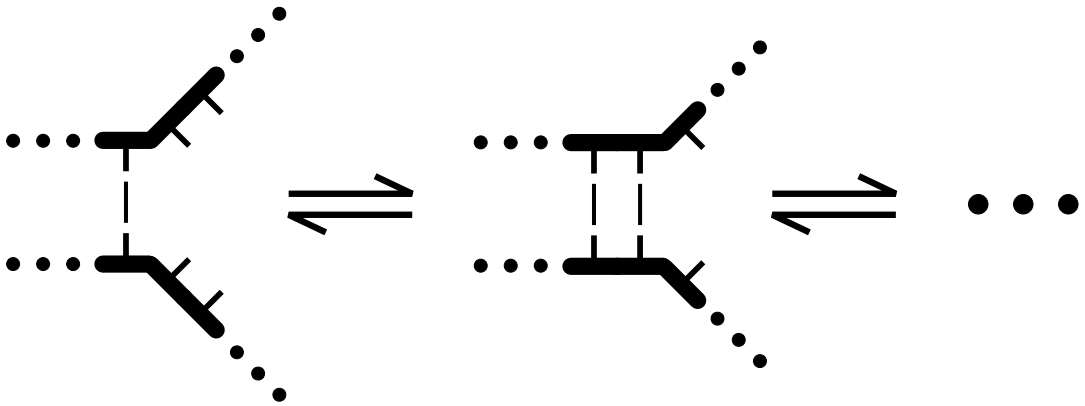


Figure 8.1: A single step of DNA zippering

We wish to calculate the average $f(\Delta G_{step})$ for a single zippering step in our model, using an equal weighting across all possible Watson-Crick base pairings. This was performed by averaging across all 4^3 base pair combinations possible for the three base pairs shown in Figure 8.1; we note that all three base pairs are necessary for the energy calculation for some choices of energy model parameters, specifically the "dangles" parameter.

The resulting average $f(\Delta G_{step})$ for zippering steps is summarized in Table 8.1 for a temperature of 37 °C, using the NUPACK energy model with a DNA substrate, under all three "dangles" parameters and for the Metropolis [12] and Kawasaki [11] rate methods.

Rate Method	Dangles "None"	Dangles "Some"	Dangles "All"
Metropolis	1.00	.935	.935
Kawasaki	3.34	4.54	4.54

Table 8.1: Average $f(\Delta G_{step})$ for forward zippering steps, using different dangles parameters and rate methods, at 37 °C

We know that the average rates for the dangles "Some" and "All" are the same. This is due to these dangles parameters predicting the exact same ΔG_{step} for the given structures.

For the Metropolis rate method, all steps where $\Delta G_{step} < 0$ (e.g., favorable) have an

energy difference factor of 1.0. Hence we might expect the Metropolis rate method to have a consistent average forward zippering rate across all dangles parameters. However this is not the case: for dangles "Some" and "All", there are 8 of the 64 distinct zippering steps which are unfavorable, and thus our average zippering rate is less than 1.0.

In Wetmur and Davidson [29], the authors state that there is an effect on zippering due to the GC content, with a 1.84-fold rate change between 34% and 64% GC content at 80 °C. In our kinetics model, the Kawasaki rate method could lead to a significant difference in zippering rate due to GC content, while the Metropolis rate method is very unlikely to do so. In the Kawasaki rate method with a temperature of 37 °C, we can get a range for $f(\Delta G_{step})$ of 0.329 to 19.8 for dangles "Some" or "All", or 1.6 to 6.16 for dangles "None". In the Metropolis rate method, all favorable zippering steps will always have $f(\Delta G_{step})$ of 1.0. For dangles "None", this is every step and so we could not predict a difference in zippering rate due to GC content. For dangles "Some" or "All", this is 56 of the 64 different zippering steps and so we would not expect a significant change in zippering rate due to GC content.

We now calculate a calibrated k_{uni} given the experimental k_{zip} and our computed $f(\Delta G_{step})$, using the equation $k_{uni} = k_{zip}/f(\Delta G_{step})$ from before. The resulting set of calibrated k_{uni} parameters, for temperatures of 25 °C and 37 °C and the same energy model parameters as above, is summarized in table 8.2.

Rate Method, Temperature	Dangles "None"	Dangles "Some" and "All"
Metropolis, 25 °C	$4.2 * 10^8$	$4.4 * 10^8$
Metropolis, 37 °C	$6.8 * 10^8$	$7.3 * 10^8$
Kawasaki, 25 °C	$9.5 * 10^7$	$6.1 * 10^7$
Kawasaki, 37 °C	$2.1 * 10^8$	$1.5 * 10^8$

Table 8.2: Calibrated k_{uni} parameters in s^{-1}

8.2 k_{bi} Calibration

We calibrate the k_{bi} parameter for DNA substrates using experimental data on DNA hybridization rates [14] for specific 10 and 20 nucleotide DNA strands. These experiments used fluorescence measurements of the hybridization reaction to determine the bimolecular

hybridization rate k_{hyb} between complementary strands in terms of the activation energy E_{hyb} and the natural log of the collision factor A_{hyb}, as related by the following equation:

$$\ln k_{hyb} = \ln A_{hyb} - E_{hyb}/RT$$

Using their experimental values for $\ln A_{hyb}$ and E_{hyb}, we compute the following table of values for the bimolecular association rate k_{hyb} for two different temperatures (25 °C and 37 °C) and the two lengths used:

Length	k_{hyb} at 25 °C	k_{hyb} at 37 °C
10-mer	$8.14 * 10^6$	$1.56 * 10^7$
20-mer	$5.97 * 10^6$	$1.74 * 10^7$

Table 8.3: Bimolecular association rate k_{hyb} in units of /M/s for varying temperatures and DNA strand lengths

In order for us to use this hybridization rate experiment to calibrate our k_{bi} parameter, we are going to need an estimate of the expected time for a hybridization reaction to complete. Given that we are going to run simulations with a single copy of each molecule in a fixed volume V, we must convert the bimolecular hybridization rate k_{hyb} into a stochastic reaction parameter c_{hyb} which can be used to calculate the propensity of the reaction in the stochastic regime. Since the reaction is bimolecular, we know k_{hyb} and c_{hyb} are related by $k_{hyb} = V * N_a * c_{hyb}$, where N_a is Avogadro's number [9]. The reaction propensity for a bimolecular reaction with two differing species A and B is $\#A * \#B * c_{hyb}$, and in our simulation we have exactly one copy of each molecule, so the reaction propensity is $1 * 1 * c_{hyb}$. From this, we can calculate the expected time for such a reaction as $t_{exp} = \frac{1}{c_{hyb}}$ since the times are exponentially distributed with the reaction propensity c_{hyb}.

We now run a set of simulations which measure the first passage times for the strands to react and form a fully complementary duplex. In order to compare against hybridization rates, we need the measured time to be dominated by the bimolecular step of the reaction. Our simulation is run using k_{uni} as calibrated in the previous section, $k_{bi}^{sim} = 1.0 * 10^3$ /M/s, and $V = \frac{1}{N_a}$, which corresponds to a mass action concentration of $1M$. Using these values the bimolecular association rate is 5 orders of magnitude slower than the unimolecular rate, and thus the simulation should be dominated by the bimolecular step. We verified that this was the case via simulation.

We can compute a calibrated k_{bi} parameter using t_{exp} and our simulation result t_{sim}, the expected time for the simulation to reach a fully hybridized state. If we are in a regime where the bimolecular association step is the rate-limiting step, we expect that scaling the bimolecular rate up by a factor of t_{scale} should lead to expected times being faster by the same factor t_{scale}. The scaling factor t_{scale} is the simulated time divided by the expected time. Under these assumptions, if we use $k_{bi}^{calibrated} = t_{scale} * k_{bi}^{sim}$ we would expect our simulated times to approximate the expected t_{exp}. As we will see in Section 8.3.2, the assumption here does not hold perfectly, and we will discuss what can be done about that.

The resulting calibrated values for k_{bi} across the same range of energy parameters and kinetic models as in the k_{uni} calibration is summarized in Table 8.4. In each case, we used the data from the 10 nucleotide hybridization system, using 1000 simulated trajectories with a maximum simulation time of 200s (which was never reached in any trajectory). Calculations were done at 37 °C and 25 °C, using the NUPACK energy model with a DNA substrate.

Rate Method, Temperature	Dangles "None"	Dangles "Some"	Dangles "All"
Metropolis, 37 °C	$8.18 * 10^5$	$1.40 * 10^6$	$1.41 * 10^6$
Kawasaki, 37 °C	$8.07 * 10^5$	$1.38 * 10^6$	$1.38 * 10^6$
Metropolis, 25 °C	$6.02 * 10^5$	$1.26 * 10^6$	$1.29 * 10^6$
Kawasaki, 25 °C	$6.07 * 10^5$	$1.29 * 10^6$	$1.28 * 10^6$

Table 8.4: Calibrated values for k_{bi} (/M/s) for varying energy model and kinetic model parameters

Here we note that the two differing rate methods have almost no effect on the calibrated k_{bi}, which is to be expected as the different rate methods only change the kinetics of unimolecular steps, and these have already been calibrated to achieve the same zippering rate to reach the full hybridization. However, there is a difference across the dangles parameters; this is also to be expected, as the energy difference due to structure (e.g., not due to ΔG_{volume} or ΔG_{assoc}) in the bimolecular step is entirely due to dangles, and is the primary factor in the bimolecular disassociation. Thus there is competition between starting the hybridization after an initiating nucleation event and immediately falling back apart. Indeed, this effect results in a nonlinearity between k_{bi} and k_{hyb} that will be discussed in Section 8.3.2.

8.3 Discussion

8.3.1 Choice of Experimental Paper for Determining k_{uni}

We chose to calibrate the unimolecular rates using the DNA zippering rate from Wetmur and Davidson [29]. This was primarily due to the zippering rate being directly analogous to the most basic steps in our kinetics model. However, we could also have chosen to calibrate the unimolecular rates using a more complicated rate, such as that for three-way branch migration in DNA [17] and indeed that is what was done in the Master's thesis [22]. For all simulations in that thesis, $k_{uni} = 1.6 * 10^6$ s^{-1} and $k_{bi} = 9.07 * 10^3$ s^{-1} were chosen so that they approximately match a three-way branch migration step time of 10 μs at 37 °C, and a bimolecular hybridization rate constant for a 10-mer of $k_{hyb} = 1.0 * 10^6$ /M/s at 37 °C.

When we tried to calibrate the k_{uni} parameter using the three-way branch migration rate found in [17] (12 μs), we found that the resulting k_{uni} was much slower than that presented above for the calibration using zippering rate, and this led to problems calibrating the k_{bi} parameter against the much faster hybridization rate found in [14]. The linear scaling assumption did not hold, namely we found that for these k_{uni} values, a linear increase in k_{bi} as suggested by calibration experiments led to a sublinear increase in the simulated bimolecular hybridization rate. We believe this is due to the k_{bi} values needed being several orders of magnitude greater than the k_{uni} calibrated from the three-way branch migration rate.

When k_{bi} is the same order of magnitude as k_{uni}, the bimolecular disassociation rate is on the same time scale as the zippering rate. While increasing k_{bi} at this stage increases the bimolecular association rate, it also decreases the probability that the duplex will hybridize before it falls apart. For discussion purposes, we could model this probability of hybridization in a simple way, by assuming that once we make one zippering step we will proceed to fully hybridization. We arrive at the probability of hybridization, $P(hybridize) = k_{zip}/(k_{zip} + k_{disassociate})$, and thus our rate of hybridization is $k_{hyb} = k_{associate} * P(hybridize)$. Thus if $k_{disassociate} >> k_{zip}$, we get $k_{hyb} = k_{zip} * k_{associate}/k_{disassociate}$. So we will never be able to get a hybridization rate that is greater than a factor of $k_{associate}/k_{disassociate}$ larger than our zippering rate (or approximately k_{uni}). Furthermore, the ratio of $k_{associate}/k_{disassociate}$ is fixed by the thermodynamic energy model as $e^{-(\Delta G(associate) - \Delta G(disassociate))/RT}$, where the energies involved are of the newly associated or disassociated state, respectively. Thus

since the three-way branch migration data led to a much lower k_{uni}, we were not able to use that data to match the experimental k_{hyb} found in Morrison and Stols [14] due to this limit.

The important observation to take away here is that with only two adjustable kinetic parameters, the thermodynamic energy landscape is not likely to account for all kinetic phenomena of interest and indeed this is the case for zippering vs three-way branch migration. Additionally, we found that in our model the zippering rate is compatible with the high hybridization rates that have been measured, while the three-way branch migration rate is compatible only with significantly lower hybridization rates.

8.3.2 Nonlinearity of k_{bi} Calibrations

Following the discussion in Section 8.2, we made the assumption that a direct scaling of k_{bi} would lead to a directly inverse scaling in the expected time t_{sim} for the simulation. This is true in the regime where a nucleation event is followed by rapid zippering to achieve a fully hybridized duplex, as the rate controlling step should entirely be the bimolecular association step.

We characterized how well our calibrated k_{bi} values actually predicted the t_{exp} we were calibrating against. As summarized in Table 8.5, we ended up with values for t_{sim} that ranged from 0.28 to 3.39 times larger than the t_{exp} which we desired. Looking at this ratio of t_{sim}/t_{exp} for the different datasets in further detail, we find that 10-mers at 37 °C average a ratio of 1.50, 10-mers at 25 °C average a ratio of 2.41, 20-mers at 37 °C average a ratio of 0.78, and 20-mers at 25 °C average a ratio of 0.44.

This is not exactly what we predicted with our direct linear scaling argument. In fact, with the values of k_{bi} and k_{uni} that we arrived at, this argument no longer holds due to the bimolecular disassociation step being within an order of magnitude of the initial zippering steps. In this regime, an additional controlling factor will by how many nucleation events are necessary before the zippering begins (once it does begin, we expect it to go to completion due to it being strongly thermodynamically favorable). Wetmur and Davidson [29] also inferred that multiple nucleation attempts would be necessary prior to zippering commencing, and thus that DNA hybridization is not diffusion-limited.

Thus our calibrated k_{bi} parameters are no longer in the regime where the rate controlling step is the bimolecular association and there must be a more complicated process involved,

Length, Rate Method, Temperature	t_{sim}:"None"	t_{sim}:"Some"	t_{sim}:"All"	t_{exp}
10-mer, Kawasaki, 37 °C	$9.50 * 10^{-5}$	$9.26 * 10^{-5}$	$8.38 * 10^{-5}$	$6.41 * 10^{-5}$
10-mer, Metropolis, 37 °C	$1.17 * 10^{-4}$	$9.76 * 10^{-5}$	$9.10 * 10^{-5}$	$6.41 * 10^{-5}$
10-mer, Kawasaki, 25 °C	$2.25 * 10^{-4}$	$4.16 * 10^{-4}$	$2.82 * 10^{-4}$	$1.23 * 10^{-4}$
10-mer, Metropolis, 25 °C	$2.56 * 10^{-4}$	$3.46 * 10^{-4}$	$2.49 * 10^{-4}$	$1.23 * 10^{-4}$
20-mer, Kawasaki, 37 °C	$5.54 * 10^{-5}$	$3.25 * 10^{-5}$	$3.06 * 10^{-5}$	$5.75 * 10^{-5}$
20-mer, Metropolis, 37 °C	$7.81 * 10^{-5}$	$3.91 * 10^{-5}$	$3.42 * 10^{-5}$	$5.75 * 10^{-5}$
20-mer, Kawasaki, 25 °C	$1.11 * 10^{-4}$	$4.77 * 10^{-5}$	$4.76 * 10^{-5}$	$1.68 * 10^{-4}$
20-mer, Metropolis, 25 °C	$1.33 * 10^{-4}$	$5.24 * 10^{-5}$	$4.62 * 10^{-5}$	$1.68 * 10^{-4}$

Table 8.5: t_{sim} values for 10- and 20-mer hybridization systems, with either the Kawasaki or Metropolis rate method, dangles "None", "Some" or "All", at a temperature of either 25 °C or 37 °C and a concentration of $1 * 10^{-3}$M. All units are in seconds.

such as the competition between bimolecular disassociation and zippering. While scaling the k_{bi} values up from our calibrated numbers gave us closer values for t_{sim} to those desired, the resulting changes were nonlinear in the scaling factor.

So while our calibrated numbers give us results to within an order of magnitude, further work is necessary in order to better understand the relationship between k_{uni} and k_{bi}, especially when considering the bimolecular disassociation events and the competition between those and initiation of zippering.

8.3.3 Other Substrates

While the references used were for DNA substrates, we note that similar numbers have been estimated for RNA substrates for both hybridization and zippering, though many of the relevant experiments are done with much larger strand lengths and thus are not suitable for direct simulation for the bimolecular calibration. Thus for calibrating against a RNA substrate we use the same experimental values as for DNA and just run the analysis (for the average zippering rate) and simulation (for calibrating k_{bi}) using the NUPACK RNA energy model.

Chapter 9

Case Study: Toehold-Mediated Four-Way Branch Migration

Let us now look at an experimental system which will we study using the Multistrand simulator. Our goal is to highlight how we can use the simulator to explore actual experimental systems and acquire detailed simulation data which is then useful for comparison or prediction of the experimental system.

9.1 Background

Toehold-mediated strand exchange is an important mechanism in the design and control of DNA nanotechnology. Toehold-mediated three-way branch migration has been used to implement many DNA devices, such as switches [25], circuits [23], motors [32], and many other motifs. The kinetics and mechanism for toehold-mediated three-way branch migration has been well characterized [35, 33] with reaction rates that can be controlled over six orders of magnitude by varying the length and/or strength of the toehold.

While four-way branch migration kinetics have been well characterized [16, 27], analogous control over the process by the use of toeholds was only recently characterized by Dabby, et al. [5]. In this section we will explore the use of Multistrand to simulate the toehold-mediated four-way branch migration and compare with the experimental results.

9.2 Mechanism

The mechanism involved in toehold-mediated four-way branch migration is shown here in Figure 9.1. The mechanism begins with the toehold binding, where the domains **m** and **n**

bind to their respective complements $\mathbf{m}^*$ and $\mathbf{n}^*$. The formation of these toeholds stabilizes the resulting complex, so that it remains bound together for a reasonable timescale. A four-way branch migration can then occur. Four-way branch migration whereby the branch migration regions (domains $\mathbf{x}$ and $\mathbf{x}^*$ in the respective complexes) exchange via a random walk, and allow the eventual disassociation into two fully paired DNA duplexes. While the entire process is theoretically reversible, this final state is typically more stable than the starting state due to the additional bonding energy from the toehold regions.

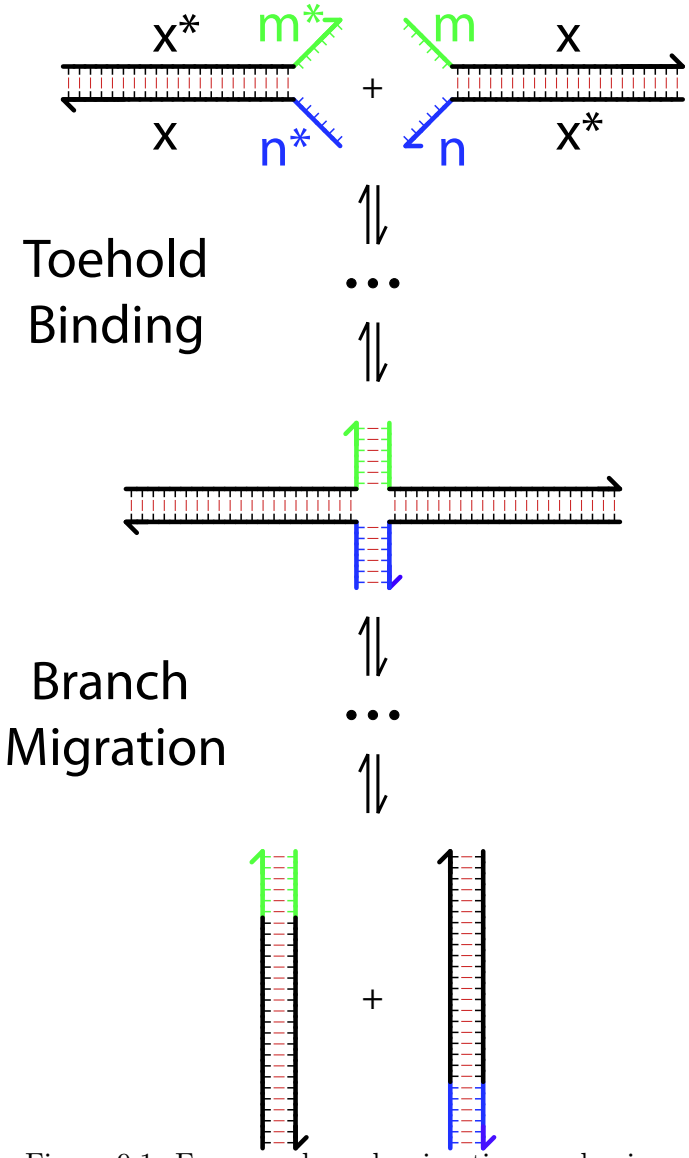


Figure 9.1: Four-way branch migration mechanism

The primary means of controlling this mechanism in DNA nanotechnology is by changing the lengths of the toehold domains. In the experiments conducted by [5], the toehold

domains **m** and **n** vary in length from 0 to 6, while the complementary domains $\mathbf{m}^*$ and $\mathbf{n}^*$ are fixed at a length of 6. This reduction in toehold length (truncated from the $5'$ side of **m** and $3'$ side of **n**) reduces the stability of the intermediates, thus increasing the likelihood of disassociating back into the starting configuration instead of proceeding with the branch migration to completion. For very short toehold domains (length less than 2), the collision intermediate will not have a stabilized four-way branched multiloop. This leads to additional slowdown in the overall process due to the instability of the complex during the initial step(s) of the branch migration.

9.3 Simulation

There are two rates that are experimentally measured in this system [5], illustrated here in Figure 9.2. These are $k_1(m, n)$, the bimolecular rate constant for successful reactions, and $k_2(m, n)$, the unimolecular rate constant for a successful reaction. Both of these rate constants are dependent on the choice of toehold regions **m** and **n**. Here, we are most interested in $k_1(m, n)$, as the bimolecular reaction will be the controlling step in the mechanism for most experimental conditions.

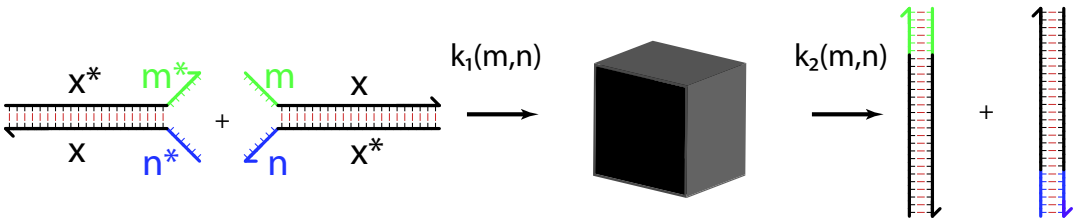


Figure 9.2: The experimentally measured rates: $k_1(m, n)$ is the bimolecular rate constant for a successful reaction, and $k_2(m, n)$ the unimolecular rate constant for a successful reaction. The intermediate state is shown as a black box, as these rate constants are for successful reactions and thus there is no well-defined intermediate which is guaranteed to always lead to a successful reaction.

To simulate this mechanism, we use first step mode (as described in Section 7.6). This allows us to acquire data for $k_1(m, n)$ and $k_2(m, n)$ directly. These rate constants, respectively, correspond to the bimolecular rate of successful reactions (k_1) and unimolecular rate of successful reactions (k_2) found in the first step model (see equation 7.6 and related discussion).

We also wish to gather data for $|m| = 0, |n| = 0$, where there is no direct toehold binding, but rather a possibility for the branch migration process to start spontaneously if the ends of the helix are breathing and thus unpaired. In order to correctly model this type of reaction, we require our starting state complexes to be Boltzmann sampled out of the ensemble of all possible secondary structures found when the given strands have formed a complex. Since the representative starting state complexes (shown in Figure 9.3B) are highly stable, most Boltzmann sampled structures are going to be very similar but with some base pair breathing.

This allows the possibility that there exists a bimolecular step from the starting state. When the Boltzmann sampling gives us a starting state where there are no bimolecular steps, the resulting trajectory is a failed collision with a 0.0 collision rate.

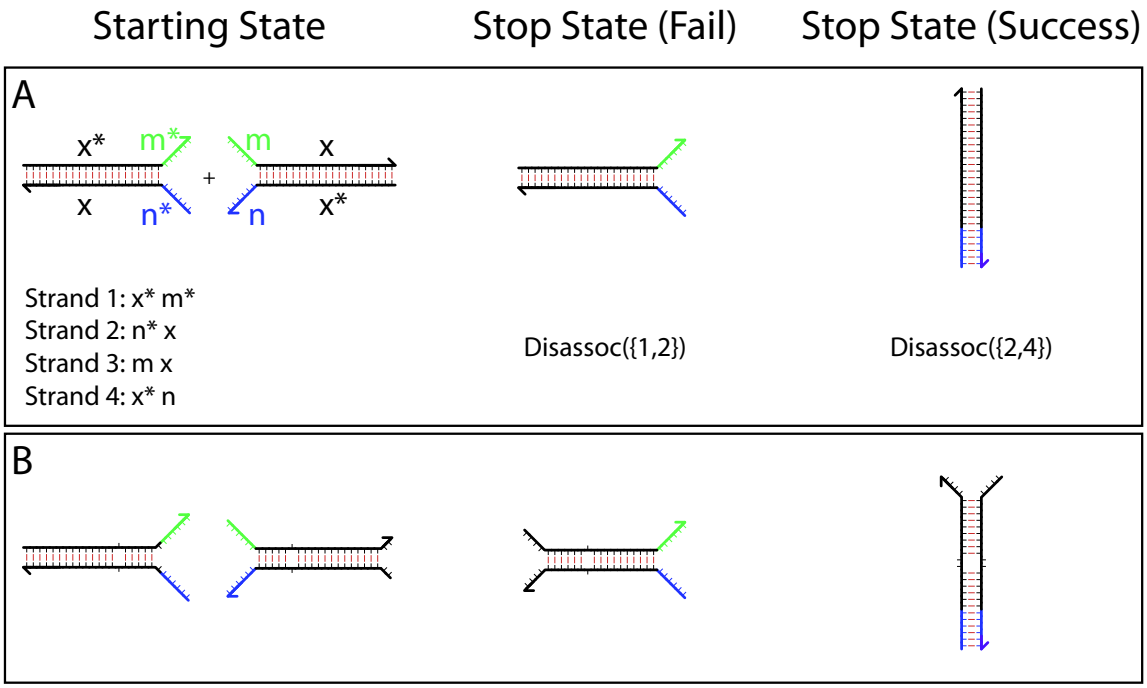


Figure 9.3: Start and stop states for the simulator. Group A shows the most energetically favorable structures to occur in the start state (due to Boltzmann sampling) or the stop state (due to the Disassoc macrostate). Group B shows a sampled structure from those that could occur in valid starting or stopping states.

Our simulation was run using the following sequences (Table 9.1), the NUPACK DNA parameter set with "dangles" set to "Some" (the default), and the Kawasaki rate method at a temperature of 25 °C. The simulation time was set to a maximum of 50.0 simulated seconds, and was never reached in any trajectory. These simulations used $k_{bi} = 9.07 * 10^3$

and $k_{uni} = 1.6 * 10^6$, the calibrated values used in the Master's thesis, rather than those discussed in section 8.

Domain	Sequence
m	GTGGGT
m*	ACCCAC
n	GTTAGC
n*	GCTAAC
x	GCGACACCGTGGACGTGCGGT
x*	ACCGCACGTCCACGGTGTCGC

Table 9.1: Sequences used for four-way branch migration domains, in $5'$ to $3'$ order. Domains **m** and **n** are variable length; for toehold lengths under 6, **m** is truncated on the $5'$ side and **n** is truncated on the $3'$ side.

The number of trajectories simulated varied with toehold length. These trajectories were broken into a number of data sets in order to estimate the standard error of the mean for k_1. These choices are summarized in Table 9.2. For some of these choices we were not able to get any successful trajectories. In these cases we can get an upper bound estimate on k_1 if we make the assumption that the next trajectory that would have been run would be a success, thus modifying equation 7.9 to be the following:

$$k_1 \leq \frac{1}{1 + N_{fail}} * k_{coll} \tag{9.1}$$

We note that since this upper bound estimate still depends on the average collision rate, k_{coll}, it may lead to different estimates even when the number of trajectories is the same due to variance in the collision rate. For example, with $|m| = 0, |n| = 0$, if the Boltzmann sampled starting state does not have a base pair breathing at an end of the helix, the resulting collision rate for that sampled trajectory is 0.0, and so we would expect the overall average collision rate k_{coll} to be much lower than that for $|m| = 2, |n| = 0$.

After simulation, the resulting trajectory data was loaded and used to calculate k_1 and k_2 values as described in Section 7.6. These results are summarized in Table 9.2 and the k_1 values are shown in Figure 9.4.

$\|\mathbf{m}\|$	$\|\mathbf{n}\|$	$k_1(m,n)$ (/M/s)	$k_2(m,n)$ (/s)	Total Trajectories
0	0	$< 1.97 * 10^0$		100×1000
0	2	$< 7.17 * 10^0$		100×1000
0	4	$< 1.02 * 10^1$		100×1000
0	6	$4.75 * 10^2 \pm 8.00 * 10^1$	$1.90 * 10^2 \pm 3.53 * 10^1$	100×1000
2	0	$< 7.01 * 10^0$		100×1000
2	2	$< 1.22 * 10^1$		100×1000
2	4	$4.58 * 10^1 \pm 2.60 * 10^1$	$1.54 * 10^2 \pm 5.78 * 10^1$	100×1000
2	6	$3.70 * 10^3 \pm 2.34 * 10^3$	$3.36 * 10^1 \pm 6.65 * 10^0$	10×1000
4	0	$1.30 * 10^1 \pm 1.29 * 10^1$	$4.51 * 10^2 \pm 0.00 * 10^0$	100×1000
4	2	$1.84 * 10^3 \pm 2.09 * 10^2$	$1.33 * 10^2 \pm 1.27 * 10^1$	100×1000
4	4	$1.04 * 10^5 \pm 1.26 * 10^4$	$1.24 * 10^2 \pm 2.89 * 10^1$	10×1000
4	6	$3.32 * 10^6 \pm 4.11 * 10^5$	$9.10 * 10^1 \pm 2.92 * 10^1$	10×20
6	0	$1.46 * 10^2 \pm 4.95 * 10^1$	$2.02 * 10^2 \pm 6.64 * 10^1$	100×1000
6	2	$5.17 * 10^4 \pm 1.28 * 10^4$	$6.54 * 10^1 \pm 7.20 * 10^0$	10×1000
6	4	$3.18 * 10^6 \pm 3.85 * 10^5$	$5.90 * 10^1 \pm 1.13 * 10^1$	10×20
6	6	$9.26 * 10^6 \pm 4.18 * 10^5$	$3.55 * 10^1 \pm 5.20 * 10^0$	10×10

Table 9.2: Simulation results for the rate constants k_1 and k_2 over varying toehold lengths. Total trajectories used for each combination is given as # datasets $\times$ # trajectories per dataset.

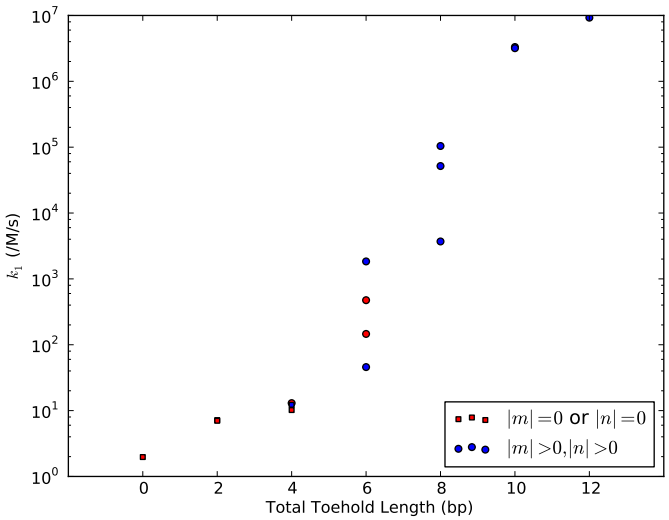


Figure 9.4: Bimolecular success rate constant k_1 vs the total toehold length $|m|+|n|$. Data points with no successful trajectories are drawn with squares instead of circles and are upper bounds on the k_1.

9.4 Discussion

We now would like to compare our simulation results against the experimental results in Dabby et al. [5]. We assume there must be a uniform scaling factor k_{scale} applied to our

$k_1(m,n)$ values in order to match the experimentally fit values $k_1^{fit}(m,n)$. We compute this scaling factor by taking the geometric mean of $k_1^{fit}(m,n)/k_1(m,n)$ over all m,n values where we had at least one successful trajectory. This leads to a $k_{scale} = 1/20.0$, and so we then compute $k_1^{sim}(m,n) = k_{scale} * k_1(m,n)$. We compare the resulting $k_1^{sim}(m,n)$ values with $k_1^{fit}(m,n)$ in Figure 9.5.

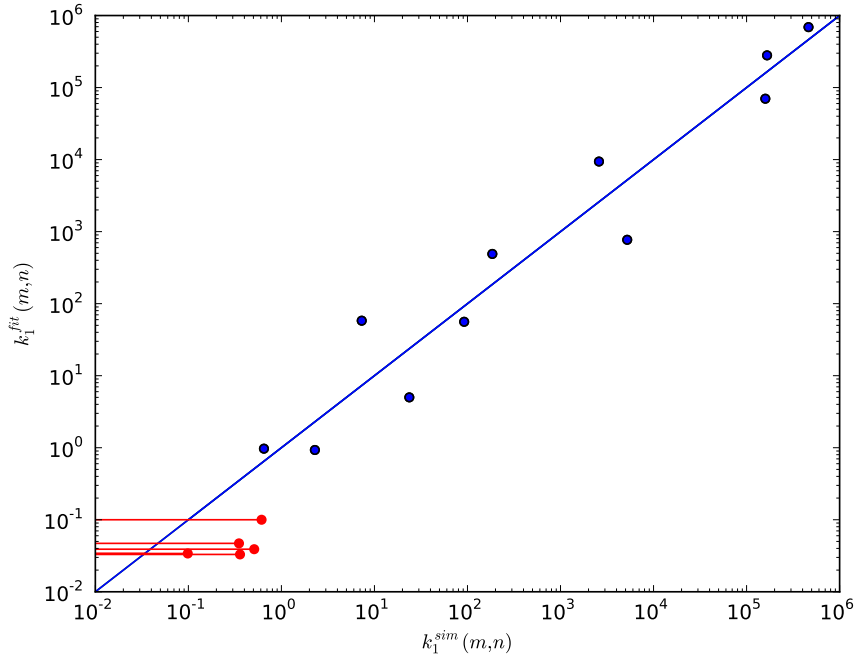


Figure 9.5: Scatter plot of $k_1^{sim}(m,n)$ vs $k_1^{fit}(m,n)$ on a log-log scale. Data points in red had only upper bounds for $k_1^{sim}(m,n)$, and are shown with a line indicating the possible range in k_1^{sim} for those m,n values.

We note that the necessity of this scaling factor assumption in order to match the experimental data is likely due to calibration factors. However, since it suggests that our k_{bi} (since that is the relevant kinetic parameter for the k_1 values) needs to be **slower** by a factor of 20.0, it is not explained by the nonlinearity in calibrating k_{bi} for DNA hybridization as discussed in Section 8.3.2.

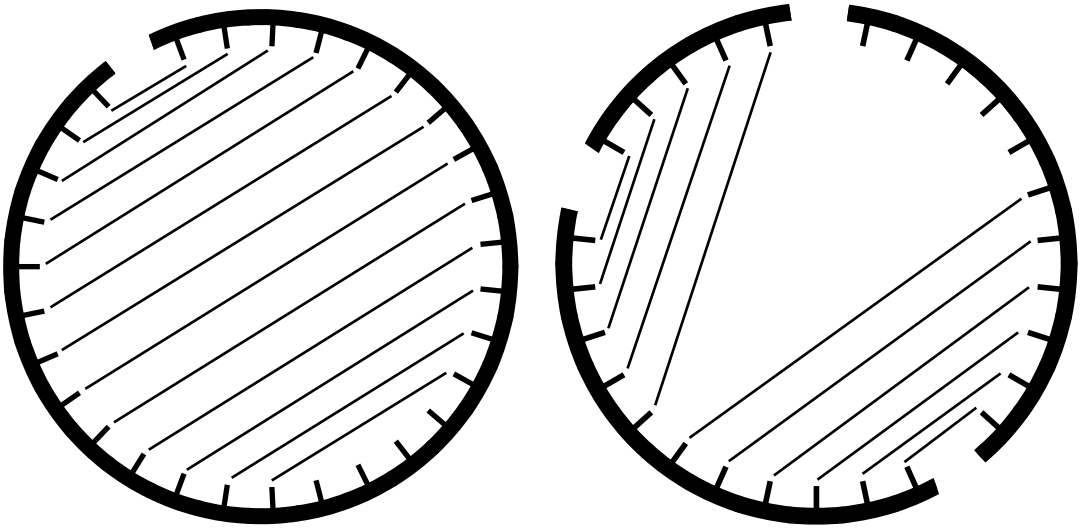
Appendix A

Strand Orderings for Pseudoknot-Free Representations

Given a complex microstate c, we can draw a *polymer graph* representation (also called the *circle-chord* representation) by laying out the strands on a circle, in the ordering $\pi^*(c)$, and representing the base pairs by chords connecting the appropriate locations (Figure A.1). In the case of a complex microstate with a single strand, we call the secondary structure *pseudoknotted* if there are crossing chords. However, the case where a complex microstate contains multiple strands requires a slightly more complex definition. We note that in this case, a strand ordering $\pi^*(c)$ corresponds to one particular way of arranging the strands on the circle; the circular permutations of $\pi^*(c)$ are the $(L-1)!$ permutations of the strands which are distinct when arranged on a circle, e.g., for three strands labeled A, B, C, there are only two distinct circular permutations: (A, B, C) and (A, C, B). With that in mind, we call an arbitrary secondary structure pseudoknotted if every circular permutation of the strand ordering has a polymer graph representation that contains a crossed chord.

While our simulator is not constrained to using a strand ordering $\pi^*(c)$ whose polymer graph representation does not contain a crossed chord, it is convenient for us to do so, as the output representation is easier to generate in these cases. The following heuristics allow us to maintain the property that our complex microstates always use a strand ordering $\pi^*(c)$ whose polymer graph does not contain a crossed chord:

The initial strand orderings we generate are based on a dot-paren structure, which naturally translate to a polymer graph with no crossed chords. The only time strand orderings can change is when performing a bimolecular move (either a break move or join move). For a break move, the resulting pair of complexes maintain the same orderings



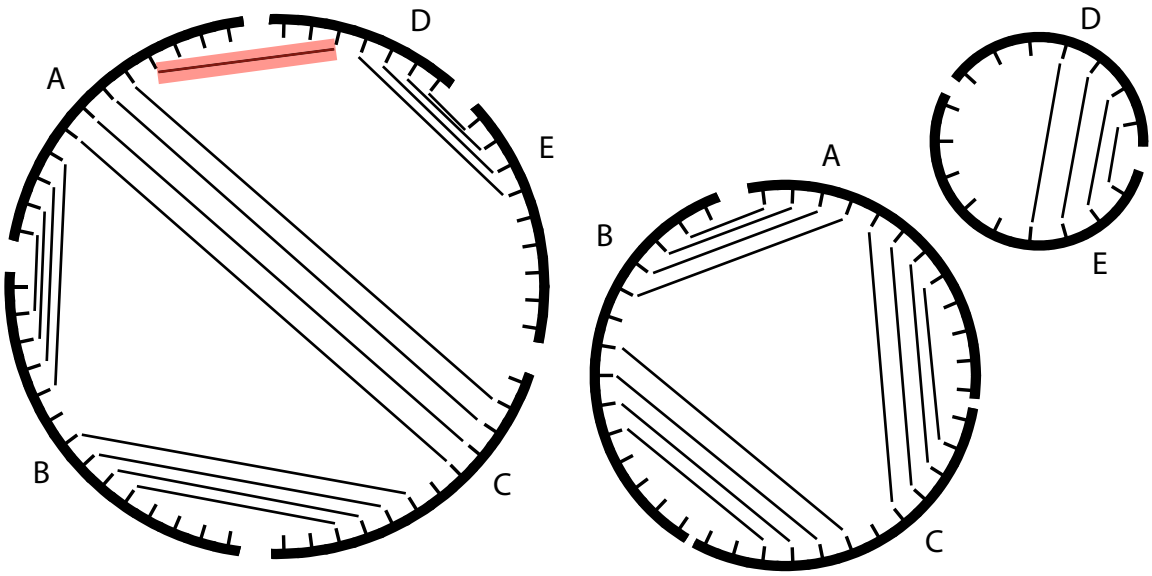
(a) A hairpin with long stem

(b) Short three-way branch migration system

Figure A.1: Two different secondary structures using polymer graph representation. Strands are always arranged with $5' \rightarrow 3'$ orientation being clockwise around the circle.

the strands originally had in the original complex—-e.g., if we had a complex of 5 strands (A through E) with ordering (A, D, E, C, B) which broke apart into two complexes with strands A, B, C and D, E, the resulting pair of orderings would be (A, C, B) and (D, E) (Figure A.2).

For a join move, we first note that a complex's open loops correspond to sequential pairs of strands in the circular strand ordering (this corresponds to the strands on either side of the "nick" in the dot-paren representation), so that a complex with ordering (A, B, D, C) has open loops corresponding to the pairs (A, B), (B, D), (D, C) and (C, A). When we perform a join move, we take each complex's strand ordering and find the cyclic permutation (these are the permutations that are identical when arranged on a circle, e.g. (A, B, C), (B, C, A) and (C, A, B) are cyclic permutations on the strand ordering (A, B, C)) which places the affected open loops at the edge of the permutation. For example, if we are joining (A, B, D, C) with (E, G, F) by the open loops around (B, D) and (E, G), we use the cyclic permutations (D, C, A, B) and (G, F, E), ending up with the strand ordering (D, C, A, B, G, F, E) for the resulting complex (Figure A.3). Why does this ordering have no crossing chords (assuming the starting ones did not)? We note that this join move was joining a base on either the $5'$ end of D or the $3'$ end of B, with a base on either the $5'$ end of G or the $3'$ end of E—in all four of these cases, the resulting chord cannot cross



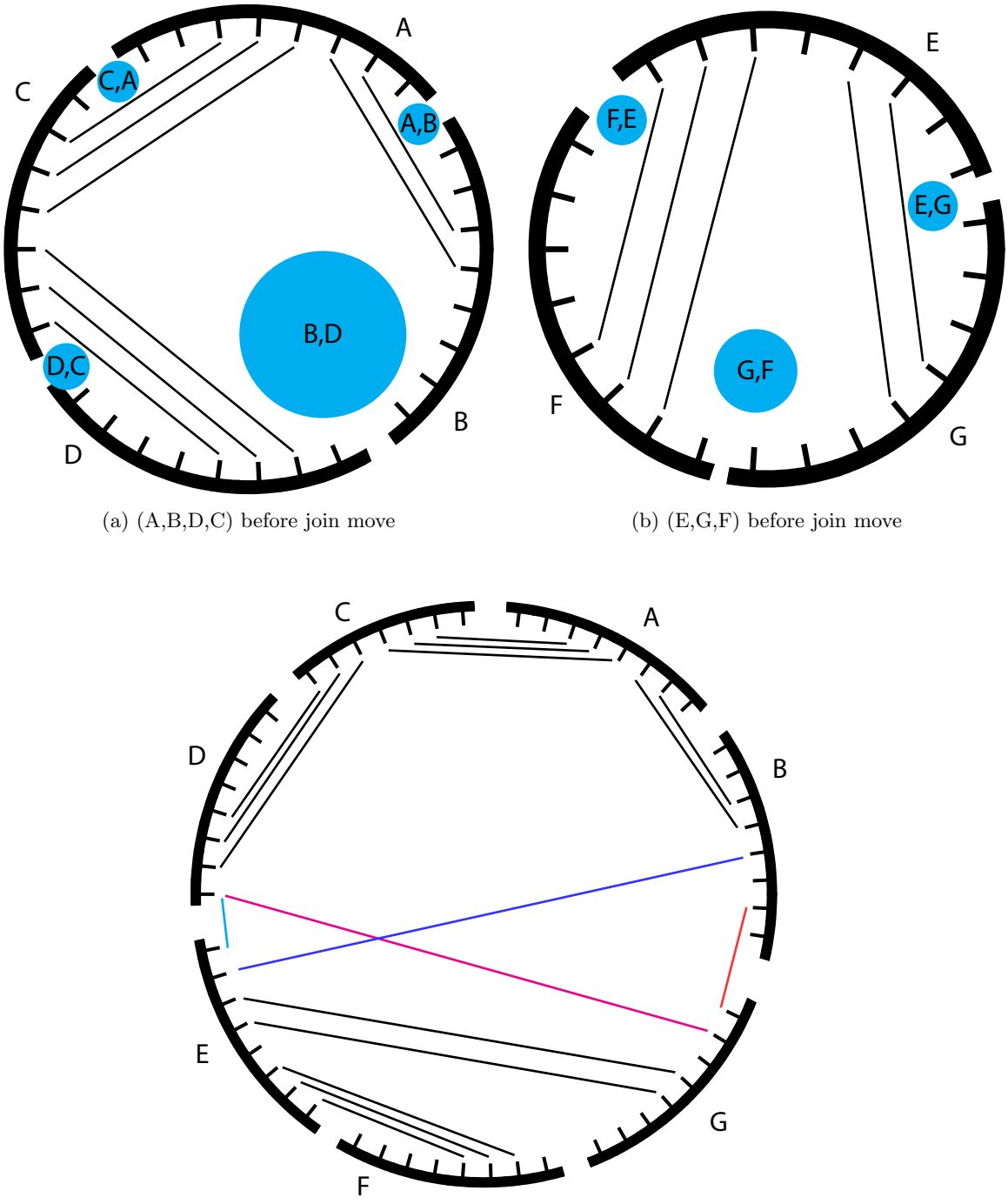
(a) (A,D,E,C,B) before break move

(b) (A,C,B) and (D,E) after break move

Figure A.2: Polymer graph representation before and after a break move (base pair highlighted in red). Note that the ordering is consistent, but we now have two separate complexes and thus two separate polymer graphs.

any existing chords if we use the given strand ordering. Note that in general, the bases available for a join move are not necessarily from the $5'$ and $3'$ edges near the nick but the same argument applies. For example, if we used a join from the (G, F) open loop (from the previous example), there is a single base on strand E that could be used to make the join, but since it's in the same open region as the $5'$ end of G and $3'$ end of F, it also could not create a crossed chord.

So we have shown that the strand ordering $\pi^*(c)$ maintained by our simulator for a complex microstate c has a polymer graph with no crossed chords. This leads naturally to the question of whether there is a different circular permutation of $\pi^*(c)$ which also has no crossed chords. The (surprising) answer is no: Every other circular permutation has at least one crossed chord! This is stated in the following theorem:



(a) (A,B,D,C) before join move

(b) (E,G,F) before join move

(c) (D,C,A,B,G,F,E) after cyclic permutations and join

Figure A.3: Polymer graph representation before and after a join move. Open loop regions are noted with a cyan circle marker. Four of the (many) possible join moves between open loops (B, D) and (E, G) are shown in (c), using red, blue, magenta, and cyan.

A.1 Representation Theorem

For every non-pseudoknotted complex microstate c, there is exactly one circular permutation $\pi^*(c)$ *whose polymer graph has no crossed chords.*

While the above heuristic can be expanded on to prove this theorem via induction on the number of strands in a complex, a more thorough proof can be found in [6] and so we will not reproduce it here.

Bibliography

[1] Jonathan Bath, Simon J. Green, and Andrew J. Turberfield. A free-running DNA motor powered by a nicking enzyme. *Angewandte Chemie International Edition*, 44(28):4358–4361, 2005.

[2] Chris Berlind. California Institute of Technology, Computer Science Senior Thesis, 2011.

[3] Victor A. Bloomfield, Donald M. Crothers, and Ignacio Tinoco, Jr. *Nucleic Acids: Structures, Properties, and Functions*. University Science Books, Sausalito, CA, 2000.

[4] Yi Chen, Mingsheng Wang, and Chengde Mao. An autonomous DNA nanomotor powered by a DNA enzyme. *Angewandte Chemie International Edition*, 43(27):3554–3557, 2004.

[5] Nadine L. Dabby, Ho-Lin Chen, Joseph M. Schaeffer, and Erik Winfree. The kinetics of toehold-mediated four-way branch migration. *Nucleic Acids Research (submitted)*, 2012.

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

[7] Robert M. Dirks, Milo Lin, Erik Winfree, and Niles A. Pierce. Paradigms for computational nucleic acid design. *Nucleic Acids Research*, 32(4):1392–1403, 2004.

[8] Christoph Flamm, Walter Fontana, Ivo L. Hofacker, and Peter Schuster. RNA folding at elementary step resolution. *RNA*, 6:325–338, 2000.

[9] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J Phys Chem*, 81(25):2340–2361, 1977.

[10] Ivo L. Hofacker. Vienna RNA secondary structure server. *Nucleic Acids Research*, 31(13):3429–3431, 2003.

[11] Kyozi Kawasaki. Diffusion constants near the critical point for time-dependent ising models. *Phys Rev*, 145:224–230, 1966.

[12] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *J Chem Phys*, 21:1087–1092, 1953.

[13] Steven R. Morgan and Paul G. Higgs. Barrier heights between ground states in a model of RNA secondary structure. *Journal of Physics A: Mathematical and General*, 31(14):3153, 1998.

[14] Larry E. Morrison and Lucy M. Stols. Sensitive fluorescence-based thermodynamic and kinetic measurements of dna hybridization in solution. *Biochemistry*, 32(12):3095–3104, 1993. PMID: 8457571.

[15] Jonathan A. Othmer. *Algorithms for mapping nucleic acid free energy landscapes*. PhD thesis, California Institute of Technology, 2009.

[16] Igor G. Panyutin and Peggy Hsieh. The kinetics of spontaneous DNA branch migration. *Proceedings of the National Academy of Sciences*, 91(6):2021–2025, 1994.

[17] Charles M. Radding, Kenneth L. Beattie, William K. Holloman, and Roger C. Wiegand. Uptake of homologous single-stranded fragments by superhelical dna: Iv. branch migration. *Journal of Molecular Biology*, 116(4):825–839, 1977.

[18] Paul W. K Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA sierpinski triangles. *PLoS Biol*, 2(12):e424, 12 2004.

[19] John SantaLucia. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proceedings of the National Academy of Sciences*, 95(4):1460–1465, 1998.

[20] John SantaLucia, Hatim T. Allawi, and P. Ananda Seneviratne. Improved nearest-neighbor parameters for predicting DNA duplex stability. *Biochemistry*, 35(11):3555–3562, 1996.

[21] John SantaLucia and Donald Hicks. The thermodynamics of DNA structural motifs. *Annual Review of Biophysics and Biomolecular Structure*, 33(1):415–440, 2004.

[22] Joseph M. Schaeffer. The multistrand simulator: Stochastic simulation of the kinetics of multiple interacting DNA strands. Master's thesis, California Institute of Technology, 2012.

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

[24] Friedrich C. Simmel. Processive motion of bipedal DNA walkers. *ChemPhysChem*, 10(15):2593–2597, 2009.

[25] Friedrich C. Simmel and Bernard Yurke. A DNA-based molecular device switchable between three distinct mechanical states. *Applied Physics Letters*, 80:883, February 2002.

[26] Friedrich C. Simmel and Wendy U. Dittmer. DNA nanodevices. *Small*, 1(3):284–299, 2005.

[27] Betty J. Thompson, Merrill N. Camien, and Robert C. Warner. Kinetics of branch migration in double-stranded DNA. *Proceedings of the National Academy of Sciences*, 73(7):2299–2303, 1976.

[28] James G. Wetmur. Hybridization and renaturation kinetics of nucleic acids. *Annual Review of Biophysics and Bioengineering*, 5(1):337–361, 1976.

[29] James G. Wetmur and Norman Davidson. Kinetics of renaturation of DNA. *Journal of Molecular Biology*, 31:349–370, 1968.

[30] Darren J. Wilkinson. *Stochastic Dynamical Systems*, pages 359–375. John Wiley & Sons, Ltd, 2011.

[31] Erik Winfree. Algorithmic self-assembly of DNA: Theoretical motivations and 2D assembly experiments. *Journal of Biomolecular Structure and Dynamics*, 11(2):263–270, 2000.

[32] Peng Yin, Harry M.T. Choi, Colby R. Calvert, and Niles A. Pierce. Programming biomolecular self-assembly pathways. *Nature*, 451(7176):318–322, 2008.

[33] Bernard Yurke and Allen Mills. Using dna to power nanostructures. *Genetic Programming and Evolvable Machines*, 4:111–122, 2003. 10.1023/A:1023928811651.

[34] Joseph N. Zadeh, Conrad D. Steenberg, Justin S. Bois, Brian R. Wolfe, Marshall B. Pierce, Asif R. Khan, Robert M. Dirks, and Niles A. Pierce. NUPACK: Analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, 32(1):170–173, 2011.

[35] David Yu Zhang and Erik Winfree. Control of DNA strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society*, 131(47):17303–17314, 2009.

[36] Wenbing Zhang and Shi-Jie Chen. RNA hairpin-folding kinetics. *Proceedings of the National Academy of Sciences*, 99(4):1931–1936, 2002.

[37] Michael Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Research*, 31(13):3406–3415, 2003.