Algorithms for computing with molecules: theory and implementation

By

Joshua Gabriel Greenspan Petrack

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

APPLIED MATHEMATICS

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Committee Member 1

Committee Member 2

Committee Member 3

Committee in Charge

2025

Contents

List of Figures

Acknowledgments

The NSF supported this thesis through grants 1844976, 1900931, and 2211793.

First, I am deeply grateful to my PhD advisor, David Doty. His research acumen and the way he treats collaboration have served as critical guideposts for me as I navigated this work and learned how to tackle problems with deeply elusive answers. I would not have been able to thrive in times of rapid discovery without David's nonchalant assurance that it's okay to be bashing your head against a wall sometimes. Most crucially, he simply treats his grad students as fellow humans. David Soloveichik has served as both a collaborator and another incredibly valuable mentor in research, and is probably the kindest person I have ever met. His attitude in welcoming me into the fold of his research group and the deep spirit of collaborative scientific inquiry that he and David Doty share is inspiring. Though I haven't had as close a relationship with Slobodan Mitrović, I'm grateful to him for serving on this thesis committee and I have enjoyed our run-ins in the realms of theory and competitive programming.

Boya Wang patiently taught me how to get started with wet lab experiments, an invaluable asset that I hope to continue making use of. Tony Szlegowski also served as a collaborator on experiments, and I'm thankful for his continued work after I left Austin. Korana Burke was a joy to work with as a TA, and my pedagogical inclinations have been shaped by the clear consideration she puts into each step of teaching. She and Qinglan Xia were very helpful in navigating teaching my own class for the first time during the beginning of the online era. Jasper Lee gave me another valuable perspective on doing theory, and his technical knowledge was remarkably timely. Matthias Koeppe also managed to give me the precise technical insight I needed at an important moment.

Much of my work has built on that of my advisor's other PhD students, and it's wonderful to have been part of such an entirely curious and congenial group. In particular, David Haley's work serves as a conceptual prerequisite for many pieces of our TBN amplifier and Eric Severson's work provided a solid foundation on which to build the implementation of our CRN simulator.

My friends in and adjacent to the math department have helped keep me level from the beginning, and I'm so glad to have had such an opportunity to get to know them as well as I have. My family has always been supportive and I do my best not to take that for granted. I'm blessed with connections that have spurred me over these years to grow into who I am now, and I could not have gotten to this point without leaning on those around me in ways big and small.

Abstract

This thesis investigates novel theoretical methods in the field of molecular computation. We investigate these ideas in theory, developing new ideas relating to existing models and rigorously establishing their correctness. But we also goes one step farther, demonstrating implementations of these ideas.

Our work encompasses two projects. The first is within the model of Thermodynamic Binding Networks (TBNs), a highly abstracted model of computation with molecules that bind along complementary sites. The model focuses purely on thermodynamics, and our focus is entirely on the question of the stability of a given configuration of molecules, rather than kinetic pathways between different configurations. We demonstrate a way to amplify signal even within these restrictions, theoretically capable of detecting a single molecule and amplifying its signal by an amount exponential in the complexity of molecules comprising the system. This system differs from common techniques like PCR due to its purely thermodynamic nature. We show this system's robustness by defining its "entropy gap", a notion of how thermodynamically unfavorable a configuration would need to be in order for the system to yield a false positive or false negative.

We then show a corresponding negative result: that within TBNs, it is impossible to amplify signal in this way more than doubly exponentially. This negative result may have more general implications on the field of thermodynamic computation, a budding research area with strong promise.

We also implement this system using DNA strands. While this implementation is a work in progress and has yet to match the exponential nature of the corresponding theory, it serves as a proof of concept that this method of amplification can work and of the TBN model more generally.

The second project concerns the model of Chemical Reaction networks (CRNs). However, rather than working within the model, this result shows an efficient algorithm for simulating stochastic CRNs, adapting work from the related but more restrictive model of population protocols. Our simulation algorithm is exact, yet still has dramatic speedup over the naïve algorithm, being able to simulate $O(\sqrt{n})$ reactions in $O(\log n)$ time on a population of n molecules, under reasonable assumptions. This bridges a gap between existing algorithms, which have not been proven to be both exact and efficient in a way that scales with molecular count.

We validate this theory by providing an implementation of this algorithm, primarily written in rust and made public via a python package called batss. While this package technically samples in approximation, we show that its output is meaningfully indistinguishable from that of exact simulation. We also show that it obtains the theoretically predicted $O(\sqrt{n})$ speedup, rendering it more efficient at large molecular counts than the most efficient existing exact algorithms. Development of this package is ongoing, and significant optimizations have yet to be made.

CHAPTER 1

Introduction

In science as a whole, the gap between theory and application can sometimes be chasmic. This is true on many levels: epistemologically, methodologically, and even socially. One powerful exception to this is the field of DNA computing, which aims to rigorously study both the theory and application of DNA nanotechnology, and ask: what kind of computation can it perform? Computation is an incredibly broad subject, capable of encompassing not just silicon computers or even just traditionally studied models like Turing machines that are associated with the theory of computation generally. Rather, computation in its breadth captures all sorts of systems that are designed to look at something and act in some prescribed way in response to it. This mindset of breadth enables DNA computing to tackle challenges across all levels of abstraction, from high-level models that blur all but the most crucial details of a system, to empirical reality that is grounded in direct observation.

Yet, it is still very easy to silo oneself into thinking primarily at some preferred layer of abstraction. Often, the constructions of science are so complex that anyone could spend their entire life staring at a single thing and still not find all of its secrets. I aim to utilize the trend in DNA computing of creating ideas that can span these boundaries. I wish both to expound rigorous theory that expands our understanding of what kinds of things might be possible at a nanoscale, and to empirically demonstrate implementations of that theory.

As my knowledge as a mathematician is primarily theoretical, this is in some ways a tall ask. However, there is a need for people who can think at all of these layers - in some ways analogous to a full-stack software developer, but specifically able to think in ways more radically different than a front end and a back end. Because concepts within intricate scientific fields tend to exhibit that intricacy self-similarly, there can be a great deal of nuance to even the basics of each point of view from which one can view a particular system or phenomenon. To this end, our focus will not be on completely arbitrary dispersed among different topics within the field. We will focus on some

of the models that the field studies, and try to wring as much insight as we can out of our analysis at each of these levels.

1.1. Models of computation

Many projects in DNA computing focus primarily on some particular model of computation. These models are useful because they are designed to be conceived at different layers of abstraction, and to have something useful to say at each of them. This is a useful method of thinking because it can spur on scientists' ability to come up with entirely new concepts. As they say, restriction breeds creativity. By focusing on one model of computation and the aspects of nanotechnology that it foregrounds, we can learn something not just about the model itself, but also about the more fine-grained reality that it belies.

Here, we will do exactly this: focus on particular models of computation, and attempt to do something that transcends the divide of theory and practice. Success in this endeavor means pushing the envelope toward understanding. After all, without theory, it is hard to truly understand why complex processes operate in the way that they do. This is a strength of theoretical computer science, and of models of molecular computation especially: by putting a particular lens over the laws of physics and chemistry and biology, we can know not just the very mechanical "why" of their operation, but also the "how" that might allow us to generalize and develop new designs that could mimic natural phenomena. Each lesson from each point at which we view some phenomenon can be propagated to other points of view, to other related phenomena, or even to seemingly disjoint areas where a similar mindset happens to yield similar results.

Here, we will focus our attention on two such models.

1.1.1. Thermodynamic binding networks. Thermodynamics underlies so many physical phenomena that its presence can easily be taken for granted. The fact that systems tend toward an equilibrium with low energy is a powerful concept, and one that is both specific enough to formalize and broad enough to generalize and view as a kind of computation. One attempt to do this formalization within DNA computing is the model of thermodynamic binding networks [12,23]. This model is highly abstract in its most general form, washing away all distinctions between how things bind to each other or even what kind of objects are binding. Rather, it cares only about what is capable of binding to what, and which things are bound together in a given configuration.

The model is still relatively in its infancy. More generally, thermodynamic computing as a subfield of molecular computing broadly is a rich area with room not just to answer questions, but to figure out the right way to ask them. What kinds of computations can be done via only the process of letting a system reach its equilibrium? What kinds of unique advantages can these computations have? How well do these theoretical abstractions match reality? These issues may be of grave importance in the coming years, as traditional computing becomes more and more costly in terms of time, energy, and physical resources, and thermodynamic computing promises an alternative that could ameliorate these crises.

Within this model, we tackle the problem of signal amplification. This is a problem with significant practical utility, allowing for detection of pathogens, toxins, biological markers, and so on. In Chapter 2, we give a rigorous theoretical definition of what this problem entails within the model, and demonstrate a specific thermodynamic binding network which can amplify signal by an amount that is exponential in the complexity of the system (i.e. the number of distinct molecules and size of molecules). We demonstrate this system's robustness in the sense that undesired system states (false positives and negatives) are programmably unfavorable compared to the system's thermodynamic equilibrium. We further the model itself, introducing new terminology and concepts that may serve in future analysis as well. Finally, we show a negative result, providing an upper bound on how much signal amplification can be done in this context, with potential implications for the field of thermodynamic computing generally.

In Chapter 3, we demonstrate some early experimental attempts to verify the validity of this theory. We show two motifs we used for implementing thermodynamic binding networks with real DNA strands, and discuss practicalities and the relationship between theory and experiment in this context.

1.1.2. Chemical Reaction Networks. Another useful aspect of chemical computation on which to focus is the way in which molecules combine and split to form other molecules. These reactions describe how chemical systems change over time, and therefore capture much of what can be called computation within them. One very widely studied model within the field is Chemical Reaction Networks. This model is used not just by those interested in chemical computation, but also more broadly by scientists in any field where the semantics of chemical reactions adequately capture the dynamics of a system they wish to study. The model is used descriptively by such

scientists, prescriptively by theorists who are interested in what kinds of things the model can do, and is even studied extensively as its own object by theorists interested in what mathematical properties it has. The model has direct scientific untility because chemical reaction networks can be implemented by DNA strand displacement cascades [58]. Thus, advancing and understanding our knowledge of both the model generally and of individual networks is valuable in the pursuit of furthering nanotechnology.

In Chapter 4, we turn our attention not to any problems within the model, but to *simulation* of the model. Simulation is important for working scientists, who may want to verify the expected behavior of a system before spending costly time and money realizing that system with real molecules or understand the array of possible behaviors that a system they are studying could potentially exhibit. The version of chemical reaction networks that most closely describes reality is stochastic, meaning that the natural way to simulate it is to simulate one reaction at a time. This algorithm, known as the Gillespie algorithm [30], is used (sometimes with slight variation or practical optimizations) very widely. Other simulation algorithms are able to simulate multiple reactions at once, but these algorithms generally only approximate the output of the Gillespie algorithm, with or without theoretical guarantees on the accuracy of that approximation. We show that it is possible to simulate many reactions at once - on average $O(\sqrt(n))$ on a system with n molecules - without sacrificing exactness. Our work builds on a brilliant insight into the related model of population protocols [7], adapting it to the more general setting of chemical reaction networks and rigorously proving that it is still both fast and accurate in theory. This insight allows for the execution of *batches* of many reactions at the same time. Our adaptation required several novel insights in order to work within the more general chemical reaction networks model, which allows for arbitrary reaction sizes and has much less predictability when modeled in continuous time compared to population protocols. We overcome these challenges respectively by transforming a chemical reaction network into one that more closely resembles a population protocol in order to allow for a variation of the batching algorithm to work properly, and by carefully choosing how we sample inter-reaction times in such a way that we can make use of work done to obtain previous samples to obtain future samples more quickly.

While a theoretical simulation algorithm does not correspond to a real system that we might want to build out of DNA, it does have an analog: an actual implementation of the algorithm

that can be run by scientists to simulate the model. We provide such an implementation. In Chapter 5, we show empirically that our implementation actually matches all expected theoretical benchmarks: it is fast, showing exactly the expected $O(\sqrt(n))$ factor of speedup that the theory predicts. It is accurate, with its output indistinguishable from that of an exact Gillespie algorithm simulator. And finally, it is able to simulate general chemical reaction networks with properties that the existing batching algorithm cannot handle.

CHAPTER 2

Signal Amplification in Thermodynamic Binding Networks

This chapter is joint work with David Doty and David Soloveichik. It was originally published as [47].

2.1. Introduction

Detecting a small amount of some chemical signal, or analyte, is a fundamental problem in the field of chemical computation. The current state-of-the-art in nucleic acid signal amplification is the polymerase chain reaction (PCR) [53]. By using a thermal cycler, PCR repeatedly doubles the amount of the DNA strand that is present. One downside is the need for a PCR machine, which is expensive and whose operation can be time-consuming. The advantages of PCR are that it can reliably detect even a single copy of the analyte if enough doubling steps are taken, and it is fairly (though not perfectly) robust to incorrect results. Recent work in DNA nanotechnology achieves "signal amplification" through other kinetic processes involving pure (enzyme-free) DNA systems, such as hybridization chain reaction (HCR) [18], classification models implemented with DNA [39], hairpin assembly cascades [68], and "crisscross" DNA assembly [44].

Although highly efficacious, PCR and these other techniques essentially rely on *kinetic* control of chemical events, and the thermodynamic equilibria of these systems are not consistent with their desired output. Can we design a system so that, if the analyte is present, the thermodynamically most stable state of the system looks one way, and if the analyte is absent, the thermodynamically most stable state looks "very different" (e.g., many fluorophores have been separated from quenchers)? Besides answering a fundamental chemistry question, such a system is potentially more robust to false positives and negatives. It also can be simpler and cheaper to operate: for many systems, heating up the system and cooling it down slowly reaches the system's thermodynamic equilibrium.

We tackle this problem of signal detection in the formal model of Thermodynamic Binding Networks (TBNs) [12, 23]. The TBN model of chemical computation ignores kinetic and geometric

constraints in favor of focusing purely on configurations describing which molecules are bound to which other molecules. A TBN yields a set of stable configurations, the ways in which monomers (representing individual molecules, typically strands of DNA) are likely to be bound together in thermodynamic equilibrium. A TBN performs the task of signal amplification if its stable configurations, and thus the states in which it is likely to be observed at equilibrium, change dramatically in response to adding a single monomer. TBNs capture a notion of what signal amplification can look like for purely thermodynamic chemical systems, without access to a process like PCR that repeatedly changes the conditions of what is thermodynamically favorable.

This chapter asks the question: if we add a single molecule to a pre-made solution, how much can that change the solution's thermodynamic equilibrium? To make the question quantitative, we define a notion of distance between thermodynamic equilibria, and we consider scaling with respect to meaningful complexity parameters. First, we require an upper limit on the size of molecules in the solution and the analyte, as adding a single very large molecule can trivially affect the entire solution. Large molecules are also expensive to synthesize, and for natural signal detection the structure of the analyte is not under our control. Second, we require an upper limit on how many different types of molecules are in the solution, as it is expensive to synthesize new molecular species (though synthesizing many copies is more straightforward).

The main result of this chapter is the existence of a family of TBNs that amplify signal exponentially. In these TBNs, there are exponentially many free "reporter" monomers compared to the number of types of monomers and size of monomers. In the absence of the analyte, this TBN has a unique stable configuration in which all reporter monomers are bound. When a single copy of the analyte is added, the resulting TBN has a unique stable configuration in which all reporter monomers are unbound. These TBNs are parameterized by two values: the first is the amplification factor, determining how many total reporter molecules are freed. The second is a value we call the system's "entropy gap", which determines how thermodynamically unfavorable a configuration of the system would need to be in order for reporters to be spuriously unbound in the absence of the analyte (false positive) or spuriously bound in its presence (false negative).

We also show a corresponding doubly exponential upper bound on the signal amplification problem in TBNs: that given any TBN, adding a single monomer can cause at most a doubly exponential change in its stable configurations. We leave as an open question to close this gap: either

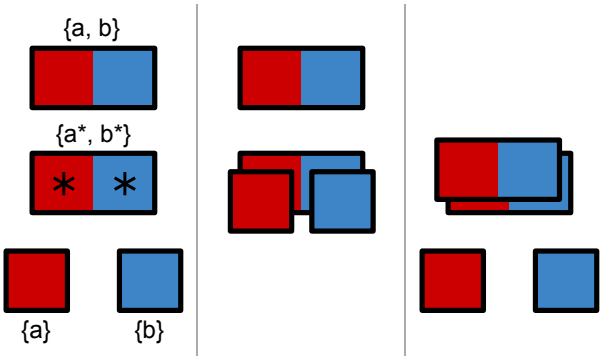


FIGURE 2.1. A simple thermodynamic binding network T with four monomers. Site types are differentiated by color. Bonds are shown by juxtaposing monomers so that unstarred sites cover starred sites. Left: the all-singleton configuration melt(T) with four polymers. Monomers are labeled by their formal identities for reference. Middle: a configuration with two polymers. As all starred sites are covered, the configuration is saturated. Right: a configuration with three polymers. As this has the most possible polymers for a saturated configuration, it is stable.

proving an exponential upper bound, or giving a TBN with a doubly-exponential amplification factor.

Our work can be compared to prior work on signal amplification that exhibits kinetic barriers. For example, in reference [44], a detected analyte serves as a seed initiating self-assembly of an arbitrarily long linear polymer. In the absence of the analyte, an unlikely kinetic pathway is required for spurious nucleation of the polymer to occur. However, in that system, false positive configurations are still thermodynamically favorable; if a critical nucleus is able to overcome the kinetic barrier and assemble, then growth of the infinite polymer is equally favorable as from the analyte. In contrast, in our system, there are *no* kinetic paths, however unlikely, that lead to an undesired yet thermodynamically favored configuration.

2.2. Definitions

2.2.1. General TBN Definitions. A *site type* is a formal symbol such as a, and has a *complementary* type, denoted a^*, with the interpretation that a binds to a^* (e.g., they could represent complementary DNA sequences). We also refer to site types as domain types, and sites as domains. We call a site type such as a an *unstarred* site type, and a^* a *starred* site type. A *monomer type* is a multiset of site types (e.g., a DNA strand consisting of several binding domains); for example monomer type $\mathbf{m} = \{a, a, a, b, c^*\}$ has three copies of site a, one of site b, and one of

site c^*. A *TBN* [11, 23] is a multiset of monomer types. We call an instance of a monomer type a *monomer* and an instance of a site type a *site*.

We take a convention that, unless otherwise specified, TBNs are *star-limiting*: for each site type, there are always at least as many sites of the unstarred type as the starred type among all monomers. Given a TBN, this can always be enforced by renaming site types to swap unstarred and starred types, which simplifies many of the definitions below.

A *configuration* of a TBN is a partition of its monomers into submultisets called *polymers*. We say that a site type (or a site) on a polymer is *uncovered* if, among the monomers in that polymer, there are more copies of the starred version of that site type than the unstarred version (otherwise *covered*). A polymer is *self-saturated* if it has no uncovered starred sites. A configuration is *saturated* if all its polymers are self-saturated. A configuration α of a TBN T is *stable* if it is saturated, and no saturated configuration of T has more polymers than α. Fig. 2.1 shows an example TBN.

An equivalent characterization of stability is in terms of merges rather than polymer counts. We say that a *merge* is the process of taking two polymers in a configuration and making a new configuration by joining them into one polymer; likewise a *split* is the process of taking one polymer in a configuration and making a new configuration by splitting it into two polymers. Maximizing the number of polymers in a saturated configuration is equivalent to minimizing the number of merges of two polymers necessary to reach a saturated configuration. To this end, some additional notation:

Definition 2.2.1. *The* distance to stability *of a saturated configuration σ is the number of (splits minus merges) necessary to get from σ to a stable configuration.*

Note that this number will be the same for any path of splits and merges, as all stable configurations have the same number of polymers.

Equivalently, distance to stability is the number of polymers in a stable configuration minus the number of polymers in σ. We only consider this value for saturated configurations to ensure it is positive and because we may interpret it as a measure of how unlikely we are to observe the network in a given state under the assumption that enthalpy matters infinitely more than entropy.

The following definitions are not restricted to saturated configurations.

Definition 2.2.2. *Given a TBN T, we say that the all-melted configuration, denoted* $\mathrm{melt}(T)$*, is the configuration in which all monomers are separate.*

Definition 2.2.3. *Given a configuration α in a TBN T, its* merginess $m(\alpha)$ *is the number of merges required to get from* $\mathrm{melt}(T)$ *to α (or equivalently, the number of monomers in T minus the number of polymers in α).*

Definition 2.2.4. *Given a configuration α in a TBN T, its* starriness $s(\alpha)$ *is the number of polymers in α which contain at least one uncovered starred site.*

We observe that α is saturated if and only if $s(\alpha) = 0$.

Definition 2.2.5. *Given configurations α and β in a TBN T, we say $\alpha \preceq \beta$ (equivalently, $\beta \succeq \alpha$) if it is possible to reach β from α solely by splitting polymers zero or more times.*

We read $\alpha \preceq \beta$ as "α splits to β". Observe that if $\alpha \succeq \beta$, then we can reach β from α in exactly $m(\beta) - m(\alpha)$ merges. In general, we may order the merges required to go from one configuration to another in whatever way allows the easiest analysis.

2.2.2. Comparing TBNs. We need some notion of how "different" two TBNs are, so that we can quantify how much a TBN changes after adding a single monomer.

Definition 2.2.6 (distance between configurations)**.** *Let α and β be two configurations of a TBN, or of two TBNs using the same monomer types. We say that the* distance $d(\alpha, \beta)$ *between them is the L^1 distance between the vectors of their polymer counts. That is, it is the sum over all types of polymers of the difference between how many copies of that polymer are in α and in β.*

Definition 2.2.7 (distance between TBNs)**.** *Given TBNs T and T', let $\mathcal{C}$ and $\mathcal{C}'$ be their stable configurations. Define the distance between T and T' as*

$$d(T, T') = \min_{\alpha \in \mathcal{C}, \alpha' \in \mathcal{C}'} d(\alpha, \alpha'). \tag{1}$$

Note that this distance is not a metric.[1] Rather, it is a way to capture how easily we can distinguish between two TBNs; even the closest stable configurations of T and T' have distance

[1]In particular, it fails to satisfy the triangle inequality, since T could have a stable configuration close to one of T', so $d(T, T') = 1$, and T' could have a different stable configuration close to one of T'', so $d(T', T'') = 1$, but T and T'' could have no close stable configurations, so $d(T, T'') > 2$.

$d(T, T')$, so we should be able to distinguish any stable configuration of one of them from all stable configurations of the other by that amount.

Note that this condition does not directly imply a stronger "experimentally verifiable" notion of distance, namely that there is some "reporter" monomer which is always bound in one TBN and always free in the other. However, the system we exhibit in this chapter does also satisfy this stronger condition. We focus on the distance given here, as it is more theoretically general and our upper bound result in Section 2.4 apply to it.

We also need a notion of how likely we are to observe a configuration of a TBN that is not stable, in order to have a notion of the system being robust to random noise. If a TBN has one stable configuration but many other configurations that are nearly stable, we would expect to observe it in those configurations frequently, meaning that in practice we may not be able to discern what the stable configuration is as easily.

We work under the assumption that enthalpy matters infinitely more than entropy, so that we may assume that only saturated configurations need to be considered. This assumption is typical for the TBN model, and can be accomplished practically by designing binding sites to be sufficiently strong. Under this paradigm, a configuration's distance to stability is a measure of how unlikely we are to observe it. This motivates the following definition:

Definition 2.2.8 (entropy gap)**.** *Given a TBN T, we say that it has an* entropy gap *of k if, for any saturated configuration α of T, one of the following is true:*

(1) α is stable.

(2) There exists some stable configuration β such that $\alpha \preceq \beta$.

(3) α has distance to stability at least k.

Note that by this definition, all TBNs trivially have an entropy gap of one. Note as well that stable configurations are technically also included in the second condition by choosing $\beta = \alpha$, but we list them separately for emphasis.

The second condition is necessary in this definition because any TBN necessarily has some configurations that have distance to stability one, simply by taking a stable configuration and arbitrarily merging two polymers together. These configurations are unavoidable but are not likely to be problematic in a practical implementation, because a polymer in such a configuration should

be able to naturally split itself without needing to interact with anything else—these configurations will never be local energy minima. Reference [11] discusses self-stabilizing TBNs in which *all* saturated configurations have this property, equivalent to an entropy gap of ∞.

2.2.3. Feed-Forward TBNs.

Definition 2.2.9. *We say that a configuration α of a TBN is* feed-forward *if there is an ordering of its polymers such that for each domain type, all polymers with an excess of unstarred instances of that domain type occur before all polymers with an excess of starred instances of that domain type.*

We say that a TBN T is feed-forward *if there is an ordering of its monomer types with this same property—that is, T is feed-forward if* $\text{melt}(T)$ *is feed-forward.*

For example, the TBN $\{(ab), (a^*c), (b^*c^*)\}$ is feed-forward with this ordering of monomers because the a, b and c come strictly before the a^*, b^* and c^* respectively. Note that not all configurations of a feed-forward TBN are necessarily feed-forward; for instance, merging the first and third monomers in this TBN gives a non-feed-forward configuration.

An equivalent characterization can be obtained by defining a directed graph on the polymers of a configuration α and drawing an edge between any two polymers that can bind to each other, from the polymer with an excess unstarred binding site to the polymer with a matching excess starred binding site (or both directions if both are possible). The configuration α is feed-forward if and only if this graph is acyclic, and the ordering of polymers can be obtained by taking a topological ordering of its vertices.

The main benefit of considering feed-forward TBNs is that we can establish a strong lower bound on the merginess of stable configurations. If *any* TBN T has n monomers that have starred sites, it will always take at least $\frac{n}{2}$ merges to cover all those sites, because each monomer must be involved in at least one merge and any merge can at most bring a pair of them together. For instance, the non-feed-forward TBN $\{\{a, b^*\}, \{a^*, b\}\}$ can be stabilized with a single merge. In feed-forward TBNs, this bound is even stronger, as there is no way to "make progress" on covering the starred sites of two different monomers at the same time.

Lemma 2.2.10. *If a configuration α is feed-forward, then any saturated configuration σ such that $\alpha \succeq \sigma$ satisfies $m(\sigma) - m(\alpha) \geq s(\alpha)$. That is, reaching σ from α requires at least $s(\alpha)$ additional merges.*

Intuitively, in a feed-forward configuration, the best we can possibly do is to cover all of the starred sites on one polymer at a time. We can never do better than this with a merge like merging $\{a, b^*\}$ and $\{a^*, b\}$ that would let two polymers cover all of each others' starred sites.

Proof. Given a feed-forward configuration α, let L be the ordered list of polymers from α being feed-forward. Partition L into separate lists (keeping the ordering from L) based on which polymers are merged together in σ. That is, for each fully merged polymer $\mathbf{P} \in \sigma$ create a list $L_{\mathbf{P}}$ of the polymers from α that are merged to form $\mathbf{P}$, and order this list based on the ordering from L. We can order the merges to reach σ from α as follows: repeatedly (arbitrarily) pick a polymer $\mathbf{P}$ from σ and merge all of the polymers in $L_{\mathbf{P}}$ together in order (merge the first two polymers in $L_{\mathbf{P}}$, then merge the third with the resulting polymer, and so on).

This sequence of merges gives us a sequence of configurations $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_\ell = \sigma$. We observe that for $1 \leq i \leq \ell - 1$, we have $s(\alpha_i) - s(\alpha_{i+1}) \leq 1$. That is, each merge can lower the starriness by at most one. We know this because each merge is merging a polymer $\mathbf{Q} \in \alpha$ with one or more other already-merged polymers from α that all come before $\mathbf{Q}$ in L. This means $\mathbf{Q}$ cannot cover any starred sites on any monomers it is merging with. The only way for the starriness of a configuration to decrease by more than 1 in a single merge is for the two merging polymers to cover all of each others' starred sites, so it follows that each merge in this sequence lowers starriness by at most 1. From this it follows that we need at least $s(\alpha)$ merges to get to σ, because $s(\sigma) = 0$.

□

Letting $\alpha = \text{melt}(T)$ (note $m(\alpha) = 0$) gives the following corollary.

Corollary 2.2.11. *Any saturated configuration σ of a feed-forward TBN T satisfies $m(\sigma) \geq s(\text{melt}(T))$.*

Because stable configurations are saturated configurations with the minimum possible merginess, this bound gives the following corollary.

Corollary 2.2.12. *If a saturated configuration σ of a feed-forward TBN T satisfies $m(\sigma) = s(\text{melt}(T))$, then σ is stable.*

2.3. Signal Amplification TBN

2.3.1. Amplification Process. In this section, we prove our main theorem. This theorem shows the existence of a TBN parameterized by two values n (the amplification factor) and k (the entropy gap). Intuitively, this TBN amplifies the signal of a single monomer by a factor of 2^n, with any configurations that give "incorrect" readings having $\Omega(k)$ distance to stability. Our proof will be constructive.

THEOREM 2.3.1. *For any integers $n \geq 1, k \geq 2$, there exists a TBN $T = T_{n,k}$ and monomer $\mathbf{a}$ (the analyte) such that if $T^{\text{a}} = T^{\text{a}}_{n,k}$ is the TBN obtained by adding one copy of $\mathbf{a}$ to $T_{n,k}$, then*

(1) T and T^{a} each have exactly one stable configuration, denoted $\sigma_{n,k}$ and $\sigma^{\text{a}}_{n,k}$ respectively, with $d(\sigma_{n,k}, \sigma^{\text{a}}_{n,k}) \geq 2^n$. In particular, there are k monomer types with 2^{n-1} copies each, with all of these monomers bound in $\sigma_{n,k}$ and unbound in $\sigma^{\text{a}}_{n,k}$.

(2) T and T^{a} each have an entropy gap of $\lfloor \frac{k}{2} \rfloor - 1$.

(3) T and T^{a} each use $O(nk)$ total monomer types, $O(nk^2)$ domain types, and $O(k^2)$ domains per monomer.

The first condition implies that $T_{n,k}$ can detect a single copy of $\mathbf{a}$ with programmable exponential strength - there is only one stable configuration either with or without $\mathbf{a}$, and they can be distinguished by an exponential number of distinct polymers. Note that this is even stronger than saying that $d(T_{n,k}, T^{\text{a}}_{n,k}) \geq 2^n$, as that statement would allow each TBN to have multiple stable configurations. The second condition implies that the system has a programmable resilience to having incorrect output, because configurations other than the unique stable ones in each case are "programmably" unstable (based on k), and thus programmably unlikely to be observed. Note that throughout this chapter we will use $\frac{k}{2}$ instead of $\lfloor \frac{k}{2} \rfloor$ for simplicity, as we are concerned mainly with asymptotic behavior. The third condition establishes that the system doesn't "cheat" - it doesn't obtain this amplification by either having an extremely large number of distinct monomers, or by having any single large monomers.

The entire TBN $T_{n,k}$ is depicted in Figures 2.2 and 2.3 with $n = 2$ and $k = 3$. The former shows the unique stable configuration before adding the analyte, and the latter shows the unique

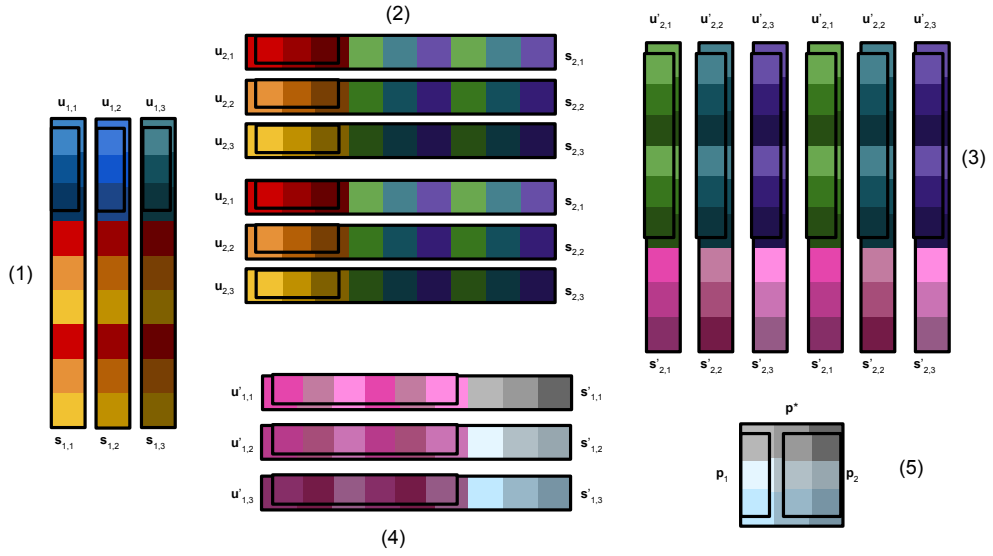


FIGURE 2.2. The unique stable configuration $\sigma_{2,3}$ of $T_{2,3}$, with 19 polymers. All starred sites are visually "covered" by unstarred sites on another monomer. The parts of the diagram are numbered by the order that the signal from the analyte will cascade through them. Parts (1) and (2) form the "first half", where the signal is doubled at each step. Parts (3) and (4) form the "second half", where the signal converges so that it can get an "entropic payoff" from part (5).

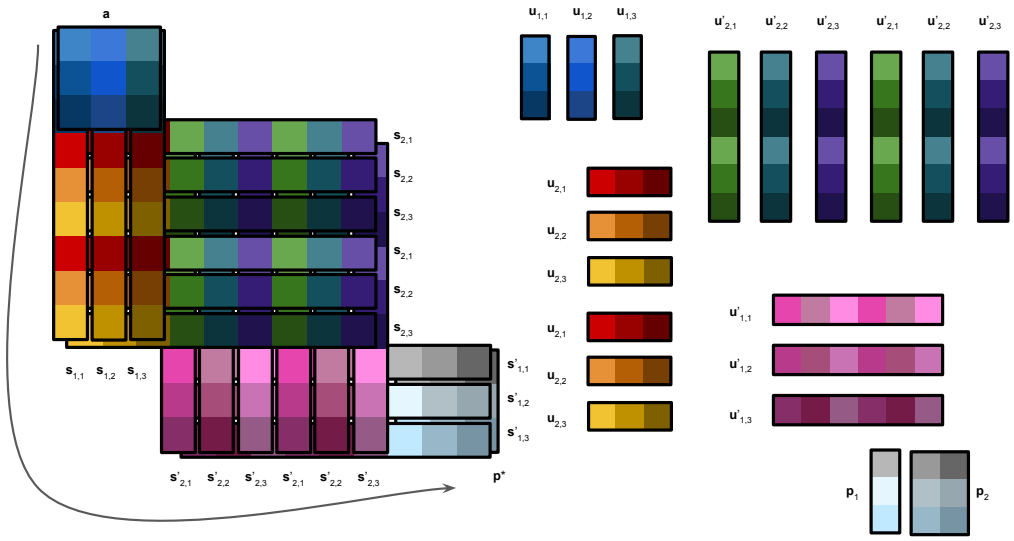


FIGURE 2.3. The unique stable configuration $\sigma^{\mathrm{a}}_{2,3}$ of $T^{\mathrm{a}}_{2,3}$. The arrow shows the conceptual order in which the analyte's signal has been propagated, with $\mathbf{a}$ covering all $\mathbf{s}_{1,j}$, which cover all $\mathbf{s}_{2,j}$, which cover all $\mathbf{s}'_{2,j}$, which cover all $\mathbf{s}'_{1,j}$, which finally cover $\mathbf{p}^*$. This configuration has 21 polymers, 2 more than $\sigma_{2,3}$: conceptually, one of these is from adding the analyte and the other is from the analyte's signal cascading through the layers to release P_1 and P_2 at the cost of one merge. As they have no polymers in common, $d(\sigma_{2,3}, \sigma^{\mathrm{a}}_{2,3}) = 19 + 21 = 40$.

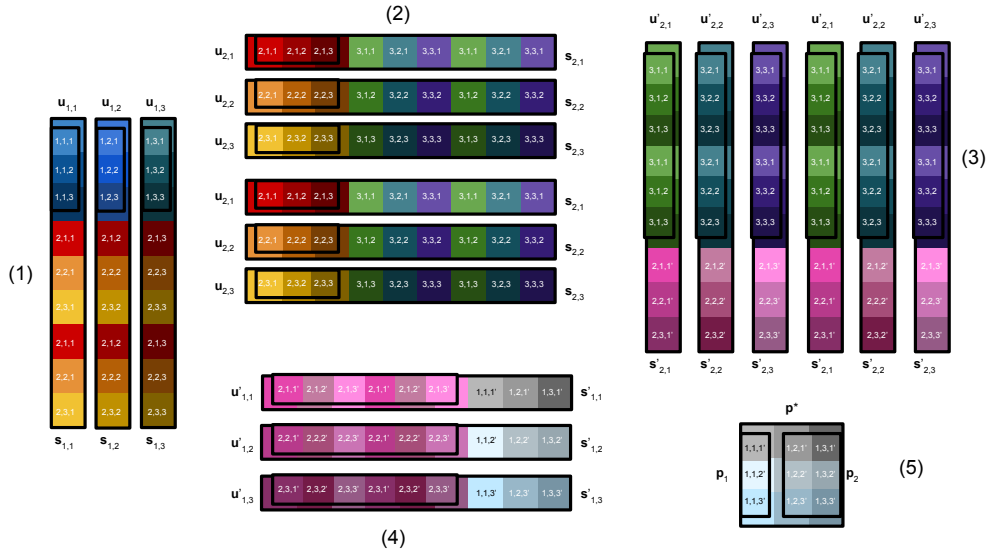


FIGURE 2.4. A version of Fig. 2.2 with text labels on domains, for accessibility and allowing comparison with the domains as defined in the text of the chapter. This figure shows the unique stable configuration of $T_{2,3}$.

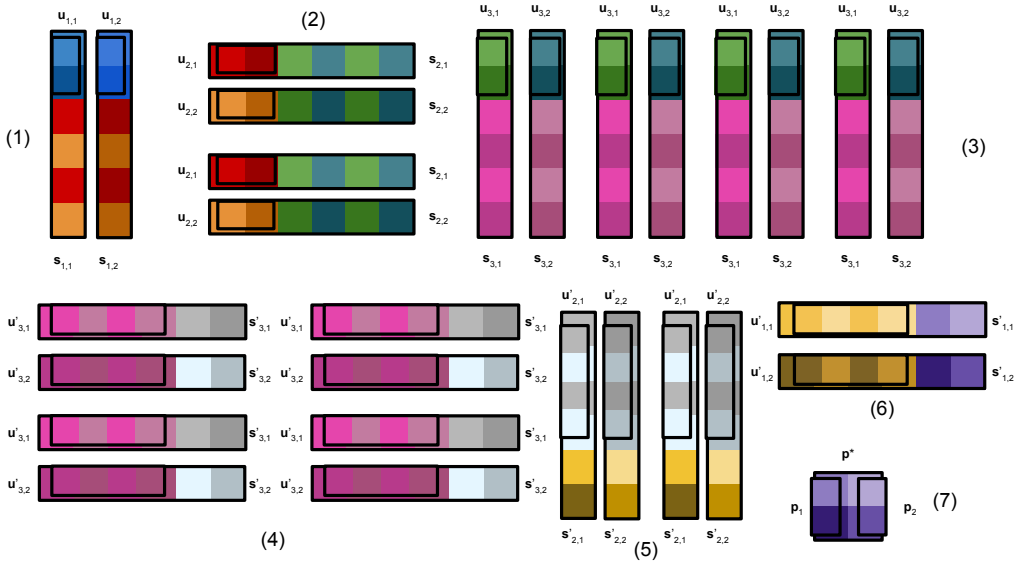


FIGURE 2.5. The unique stable configuration $\sigma_{3,2}$ of $T_{3,2}$. Compared to Fig. 2.2 (or Fig. 2.4 above), which show $T_{2,3}$, this figure shows one more layer and a smaller entropy gap parameter. The additional layer means that there are 4 copies of each monomer in the largest parts of the figure, compared to 2 copies of each monomer in the other figures; if another layer were added, it would contain 8 copies of each monomer. The smaller entropy gap parameter manifests in this figure visually having a "2 by 2 grid" design motif, compared to the "3 by 3 grid" motif in the other figures.

stable configuration after adding the analyte. For comparison, Fig. 2.5 depicting the pre-analyte configuration with $n = 3$ and $k = 2$ is shown as well.

We will start by constructing the first half of $T_{n,k}$ and describing "how it works". The monomers in this first half are the driving force that allows $T_{n,k}$ and $T^{\mathrm{a}}_{n,k}$ to have exponentially different stable configurations.

The first half of $T_{n,k}$ has monomer types $\mathbf{u}_{i,j}$ and $\mathbf{s}_{i,j}$ (named as those with only unstarred sites, and those with both starred and unstarred sites) for $1 \leq i \leq n, 1 \leq j \leq k$. It has domain types denoted as triples (i, j, ℓ) for $1 \leq i \leq n+1, 1 \leq j, \ell \leq k$. Each $\mathbf{u}_{i,j}$ monomer has k different unstarred domains, one of each (i, j, ℓ) for each $1 \leq \ell \leq k$. Each $\mathbf{s}_{i,j}$ monomer has a starred copy of each domain in $\mathbf{u}_{i,j}$, and additionally has two copies of each unstarred domain $(i+1, \ell, j)$ for each $1 \leq \ell \leq k$ (note that here the second domain type parameter varies instead of the third). For each $\mathbf{u}_{i,j}$ and $\mathbf{s}_{i,j}$ monomer, there are 2^{i-1} copies. We can conceptually break these monomers into n "layers", each consisting of all monomers with the same value for their first parameter. The analyte we wish to detect, $\mathbf{a}$, is a monomer that has one copy of each unstarred domain $(1, j, \ell)$, $1 \leq j, \ell \leq k$.

Conceptually, when the analyte is absent, the most efficient way for all starred sites on each $\mathbf{s}_{i,j}$ to be covered is by the unstarred sites on a corresponding $\mathbf{u}_{i,j}$, as seen in Fig. 2.2. Although the TBN model is purely thermodynamic, we can conceptualize that when the analyte is added, its signal can propagate "kinetically" through each layer. In the first layer, it can "displace" the k different $\mathbf{u}_{1,j}$ monomers and bind to all of the $\mathbf{s}_{1,j}$ monomers. In doing so, it brings together all the unstarred sites on all of the $\mathbf{s}_{1,j}$ monomers. Having been brought together, these sites "look like" two copies of the analyte, but with the domains from layer 2 instead of layer 1. Thus, this polymer is then able to displace *two* copies of each $\mathbf{u}_{2,j}$ from their corresponding $\mathbf{s}_{2,j}$ monomers, thus bringing all of the $\mathbf{s}_{2,j}$ together. This in turn now looks like four copies of the analyte for the domains in the third layer, and so on. Each layer allows this polymer to assimilate exponentially more $\mathbf{s}_{i,j}$, thus freeing exponentially many $\mathbf{u}_{i,j}$. Each of these displacement steps involves an equal number of splits and merges.

2.3.2. Convergence Process. So far, the TBN described has exactly one stable configuration before adding the analyte, and it performs the task of amplifying signal by having the potential to

change its state exponentially when the analyte is added. However, there is also a stable configuration after adding the analyte in which nothing else changes, and many others in which only a small amount of change occurs. We must guaranteed that the analyte's signal "propagates" through all of the layers.

To design the system to meet this requirement, we observe that all exponentially many monomers that have been brought together must contribute to some singular change in the system that gains some entropy, to spur the signal into propagating. The typical way to accomplish this in a TBN is by having monomers that have been brought together displace a larger number of monomers from some complex at the cost of a smaller number of merges. Because the pre-analyte TBN has an entropy gap of $k-1$ in this design so far, we can afford to give the TBN with the analyte an "entropic payoff" of $\frac{k}{2}$. When the analyte is absent, this payoff is weak enough that there will still be an entropy gap of $\frac{k}{2}-1$; when the analyte is present, the existence of this payoff will force the signal to fully propagate, and will give the TBN with the analyte an entropy gap of $\frac{k}{2}-1$ by making it so that any configurations in which this payoff is not achieved are also far away from stable.

Another challenge is that we cannot simply detect all our exponentially many conjoined monomers by binding them all to a single exponentially large monomer, because we need to bound the size of the largest monomer in the system. Our conceptual strategy for overcoming this is as follows: the signal will converge in much the same way as it was amplified. In the amplification step, one set of domains coming together in one layer was enough to cause two of them to come together in the next layer. In this convergence step, two sets of domains in one layer will have to converge together to activate one set in the next layer. This convergence ends in bringing together a set of binding sites that is of the same size as the analyte, which can then directly displace some monomers to gain $\frac{k}{2}$ total polymers.

We now fully define $T_{n,k}$. We start with the already described $\mathbf{u}_{i,j}$ and $\mathbf{s}_{i,j}$. To these, we first add monomer types $\mathbf{u}'_{i,j}$ and $\mathbf{s}'_{i,j}$ for $1 \leq i \leq n, 1 \leq j \leq k$. These monomers are the 'converging' equivalents of $\mathbf{u}_{i,j}$ and $\mathbf{s}_{i,j}$. Conceptually, they will activate in the reverse order: two copies of each $\mathbf{s}'_{i,j}$ for $1 \leq j \leq k$, when brought together, will be able to bring together one copy of each $\mathbf{s}'_{i-1,j}$ for $1 \leq j \leq k$.

Each $\mathbf{u}'_{i,j}$ monomer has $2k$ unstarred domains: two copies each of domains $(i+1, j, \ell)'$ for each $1 \leq \ell \leq k$. Each $\mathbf{s}'_{i,j}$ has a starred copy of each of the $2k$ domains in $\mathbf{u}'_{i,j}$ and additionally has one

unstarred domain $(i, \ell, j)'$ for $1 \leq \ell \leq k$ (note again here that the second domain type parameter varies instead of the third). One exception is the monomers $\mathbf{u}'_{n,j}$ and $\mathbf{s}'_{n,j}$ (the first ones to activate) which use domains $(n+1, j, \ell)$ and $(n+1, \ell, j)$ instead of $(n+1, j, \ell)'$ and $(n+1, \ell, j)'$ respectively so that they can interact with $\mathbf{s}_{n,j}$ monomers that have been brought together. Each monomer $\mathbf{u}'_{i,j}$ and $\mathbf{s}'_{i,j}$ has 2^{i-1} copies.

Finally, we add "payoff" monomers that will yield an entropic gain of $\frac{k}{2}$ when the signal from the analyte has cascaded through every layer. This choice of $\frac{k}{2}$ is arbitrary—a similar design works for any integer between 1 and k. Choosing a higher value leads to a higher entropy gap after adding the analyte and a lower entropy gap before adding it, and vice versa choosing a lower value. For simplicity of definitions we will assume k is even (though figures are shown with $k = 3$, which shows how to generalize to odd k).

We add one monomer $\mathbf{p}^*$, which contains the k^2 sites $(1, \ell_1, \ell_2)'^*$ for $1 \leq \ell_1, \ell_2 \leq k$. Note that this monomer can be replaced with k monomers of size k (in which case $\mathbf{a}$ would be the only monomer with more than $3k$ domains), but doing so makes the proof more complex. The idea is that when all $\mathbf{s}'_{1,j}$ monomers are already together (as they can be "for free" when $\mathbf{a}$ is present), they can cover $\mathbf{p}^*$ in one merge; if they are apart, this requires k merges. In order to make this favorable to happen when they're already together but unfavorable when they're initially apart, we add another way to cover $\mathbf{p}^*$ that takes $\frac{k}{2}$ merges. This is accomplished via monomers $\mathbf{p}_j$ for $1 \leq j \leq \frac{k}{2}$. Each $\mathbf{p}_j$ contains the $2k$ sites $(1, 2j-1, \ell)$ and $(1, 2j, \ell)$ for $1 \leq \ell \leq k$. We can interpret this geometrically as $\mathbf{p}^*$ being a square, the $\mathbf{s}'_{1,j}$ covering it by rows, and the $\mathbf{p}_j$ covering it by two columns at a time. This completes the definition of $T_{n,k}$. Recall $T^{\mathrm{a}}_{n,k}$ is $T_{n,k}$ with one added copy of $\mathbf{a}$.

Lemma 2.3.2. *$T_{n,k}$ has exactly one stable configuration $\sigma_{n,k}$.*

Proof. We consider merges to get from the melted configuration to any saturated configuration. We may order these merges such that we first make all the merges necessary to cover each individual $\mathbf{s}_{i,j}$ in increasing value of i, then each individual $\mathbf{s}'_{i,j}$ in decreasing value of i. We see that at each step of this process, we may cover the monomer in question by a single merge (of its corresponding $\mathbf{u}_{i,j}$ or $\mathbf{u}'_{i,j}$). If we never merge the corresponding $\mathbf{u}$ monomer, the only other monomers that can cover the starred sites on a given $\mathbf{s}_{i,j}$ are k different $\mathbf{s}_{i-1,\ell}$ monomers. Likewise,

the only other way to cover the starred sites on a given $\mathbf{s}'_{i,j}$ is by using k different $\mathbf{s}'_{i+1,\ell}$ monomers (except for $\mathbf{s}'_{n,j}$ which needs $\mathbf{s}_{n,\ell}$ monomers).

If an $\mathbf{s}$ or $\mathbf{s}'$ monomer is covered in multiple different ways, we order the merges such that it is first covered by one corresponding $\mathbf{u}$ monomer (and then ignore any other merges for now, as we are still ordering the merges to cover each $\mathbf{s}$ monomer sequentially). We see then that if every $\mathbf{s}$ monomer is covered by a $\mathbf{u}$ monomer, then no $\mathbf{s}$ monomers will be brought together during this process. Therefore, the first time in this sequence that we choose to cover an $\mathbf{s}$ without its corresponding $\mathbf{u}$ will require k total merges to cover that $\mathbf{s}$. The resulting configuration is feed-forward, so by Corollary 2.2.11, reaching a stable configuration requires at least one more merge per remaining $\mathbf{s}$ monomer. This results in at least $k-1$ extra merges compared to covering $\mathbf{s}$ and $\mathbf{s}'$ monomers by using $\mathbf{u}$ and $\mathbf{u}'$ monomers respectively.

Once all $\mathbf{s}$ and $\mathbf{s}'$ monomers are covered, the only other monomer with starred sites is $\mathbf{p}^*$, so we can make all the merges that are needed to cover it. If none of the $\mathbf{s}'_{1,j}$ monomers have been brought together, then the fewest merges it takes to cover $\mathbf{p}^*$ is $\frac{k}{2}$, via the $\mathbf{p}_j$ monomers. If any of them have been brought together, then it could potentially take a single merge to cover $\mathbf{p}^*$. However, this would have required $k-1$ extra merges at some point during the covering of $\mathbf{s}$ monomers, resulting in $\frac{k}{2}$ extra total merges compared to covering all $\mathbf{s}$ monomers with $\mathbf{u}$ monomers, then covering $\mathbf{p}^*$ with $\mathbf{p}_j$ monomers.

Therefore, this latter set of merges covers all starred sites in as few merges as possible, and therefore gives the unique stable configuration of $T_{n,k}$.

□

Corollary 2.3.3. *$T_{n,k}$ has an entropy gap of $\frac{k}{2}-1$.*

Proof. Recall Definition 2.2.8 for what we must show. Any saturated configuration that does not make all the merges in $\sigma_{n,k}$ must either have some $\mathbf{s}$ that is not covered by its corresponding $\mathbf{u}$ (resulting in at least $\frac{k}{2}$ extra merges, as per the above argument), or must cover $\mathbf{p}^*$ with initially-separate $\mathbf{s}'_{1,j}$ monomers (resulting in $\frac{k}{2}$ extra merges). Thus, any such configuration has distance to stability at least $\frac{k}{2}$. Any other saturated configuration that does make all of the merges in this sequence simply makes some extra merges afterward, and therefore splits to $\sigma_{n,k}$. It follows that $T_{n,k}$ has an entropy gap of $\frac{k}{2}$ (and also of $\frac{k}{2}-1$, for consistency in the statement of Theorem 2.3.1). □

Lemma 2.3.4. *$T^{\text{a}}_{n,k}$ has exactly one stable configuration $\sigma^{\text{a}}_{n,k}$, and $T^{\text{a}}_{n,k}$ has an entropy gap of $\frac{k}{2}-1$.*

PROOF. We see that $T^{\text{a}}_{n,k}$ (like $T_{n,k}$) is feed-forward (recall Definition 2.2.9) by first ordering $\mathbf{a}$ along with all the $\mathbf{u}_{i,j}$, $\mathbf{u}'_{i,j}$, and $\mathbf{p}_j$ monomers (none of which have starred sites), then all the $\mathbf{s}_{i,j}$ in increasing order of i, then all the $\mathbf{s}'_{i,j}$ in decreasing order of i, and finally $\mathbf{p}^*$.

Unlike $T_{n,k}$, however, we may reach a stable state by merging $\mathbf{a}$ together with every single $\mathbf{s}_{i,j}$, every single $\mathbf{s}'_{i,j}$ and $\mathbf{p}^*$ into a single polymer. This covers all starred sites, and requires exactly one merge per monomer with starred sites, so by Corollary 2.2.12, this configuration $\sigma^{\text{a}}_{n,k}$ is stable.

Now, we examine an arbitrary saturated configuration σ of $T^{\text{a}}_{n,k}$. We consider merges in essentially the opposite order of how they were considered when analyzing $T_{n,k}$. First, consider $\mathbf{p}^*$. It must be covered either by all the $\mathbf{p}_j$ monomers, or by all the $\mathbf{s}'_{1,j}$ monomers. If we merge all the $\mathbf{p}_j$ monomers to $\mathbf{p}^*$, we arrive at a configuration that is still feed-forward, but has only one fewer polymer with uncovered starred sites compared to $\text{melt}(T^{\text{a}}_{n,k})$ in spite of making $\frac{k}{2}$ merges. Therefore, by Lemma 2.2.10, reaching a saturated configuration from this point requires at least $\frac{k}{2}-1$ extra merges compared to $\sigma^{\text{a}}_{n,k}$.

Now, we may make a similar argument for all $\mathbf{s}$ monomers in the opposite order that we considered them in Lemma 2.3.2. First, either we have already made $\frac{k}{2}-1$ extra merges, or the $\mathbf{s}'_{1,j}$ monomers have all been brought together on a single polymer to cover $\mathbf{p}^*$. If we now make all the merges necessary to cover all starred sites on this polymer, we must do so either using all the $\mathbf{u}'_{1,j}$ or by using all the $\mathbf{s}'_{2,j}$. If we use the former, then this will require k total merges but will only reduce the count of polymers with starred sites by 1. The resulting configuration is still feed-forward, so again by Lemma 2.2.10 any saturated configuration we reach from this point will require at least $k-1$ extra merges compared to σ. Otherwise, we must bring all the $\mathbf{s}_{2,j}$ monomers together to cover these sites. This does not fall victim to the same argument, because bringing these monomers with starred sites together onto the same polymer lowers the total number of polymers with uncovered starred sites. Now that they have been brought together, the same argument shows that we must either cover all the starred sites on the $\mathbf{s}'_{2,j}$ using all the $\mathbf{s}'_{3,j}$, or suffer $k-1$ extra merges. The same argument for each layer in the converging part of the TBN also works for each layer in the amplifying part. Finally, after running through this argument we arrive at all $\mathbf{s}_{1,j}$ being brought together, which can be covered either by a single merge of $\mathbf{a}$ or by merging the k $\mathbf{u}_{1,j}$ to it.

Overall, this shows that any saturated configuration of $T^{\mathrm{a}}_{n,k}$ either makes all of the merges in $\sigma^{\mathrm{a}}_{n,k}$ or it must make at least $\frac{k}{2}-1$ extra merges. It follows that $\sigma^{\mathrm{a}}_{n,k}$ is the unique stable configuration of $T^{\mathrm{a}}_{n,k}$, with an entropy gap of $\frac{k}{2}-1$ as desired. □

These results together complete the proof of Theorem 2.3.1: each of the more than 2^n $\mathbf{u}$ and $\mathbf{u}'$ monomers (which serve as reporters) are bound in $\sigma_{n,k}$ and unbound in $\sigma^{\mathrm{a}}_{n,k}$, implying their distance is more than 2^n. The largest monomer is $\mathbf{a}$ with k^2 domains, and there are $(2n+1)k^2$ domain types and $4nk$ monomer types for the $\mathbf{s}_{i,j}$, $\mathbf{u}_{i,j}$, $\mathbf{s}'_{i,j}$, and $\mathbf{u}'_{i,j}$, plus $2+\frac{k}{2}$ more for $\mathbf{a}$, $\mathbf{p}^*$, and $\mathbf{p}_j$.

2.3.3. Avoiding Large Polymer Formation. The TBN $T^{\mathrm{a}}_{n,k}$ will, in the process of amplifying the signal of the analyte, form a single polymer of exponential size. This isn't an issue in the theoretical TBN model, but it is a practical issue because there is no way to design these monomers so that this large polymer would form.[2]

This can be solved by adding "translator gadgets". These gadgets' job is to mediate between consecutive layers. Instead of monomers from one layer directly binding to monomers from the next layer, they can split apart these translator gadgets with half of the gadget going to each layer. In exchange, the TBN will no longer have exactly one stable configuration when the analyte is present, as in the TBN model, the use of these translator gates will be purely "optional".

We define a new TBN $\widetilde{T}_{n,k}$ (as well as $\widetilde{T}^{\mathrm{a}}_{n,k}$, which is obtained by adding the analyte $\mathbf{a}$). We start with the TBN $T_{n,k}$. To assist with the amplification step, we add monomer types $\mathbf{g}_i$ and $\mathbf{g}^*_i$ for each $2 \le i \le n$. Each $\mathbf{g}_i$ consists of one copy of each unstarred domain (i,j,ℓ) for each $1 \le j,\ell \le k$. Each $\mathbf{g}^*_i$ consists of the same domains but all starred. Each of these monomers has 2^{i-1} copies. The use of these gadgets can be seen in Fig. 2.6.

To assist with the convergence step, we add monomer types $\mathbf{h}_i$ and $\mathbf{h}^*_i$ for each $2 \le i \le n+1$. Each $\mathbf{h}_i$ has *two* copies of each unstarred domain $(i,j,\ell)'$ for each $1 \le j,\ell \le k$. Each $\mathbf{h}^*_i$ has only one copy of each of the corresponding starred domains. There are 2^{i-1} copies of each $\mathbf{h}_i$ and 2^i copies of each $\mathbf{h}^*_i$. The use of these gadgets can be seen in Fig. 2.7.

THEOREM 2.3.5. *Let $\widetilde{T} = \widetilde{T}_{n,k}$ and $\widetilde{T}^{\mathrm{a}} = \widetilde{T}^{\mathrm{a}}_{n,k}$ be as described. Then:*

[2]The binding graph of the monomers within this giant polymer contains many complete k-ary trees of depth n as subgraphs. If each of the nodes of this graph is a real molecule that takes up some volume, it will be impossible to embed the whole graph within 3-dimensional space as n grows.

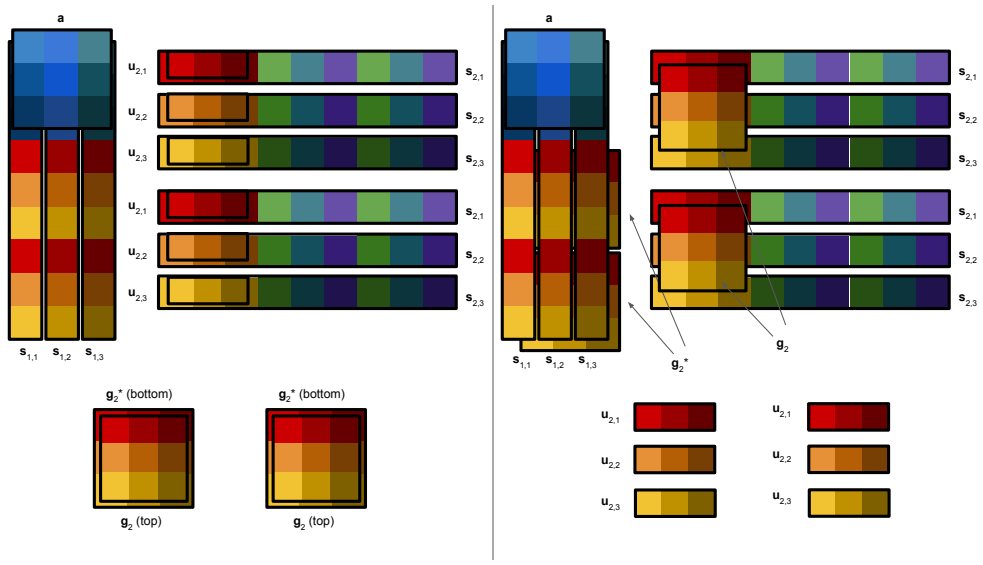


FIGURE 2.6. The amplifying translator gadget, before and after it is triggered to propagate the signal forward by one layer. When $\mathbf{s}_{1,1}$, $\mathbf{s}_{1,2}$ and $\mathbf{s}_{1,3}$ have been brought together, instead of directly replacing all the $\mathbf{u}_{2,j}$ monomers, they can split two $\{\mathbf{g}_2, \mathbf{g}_2^*\}$ complexes, and the $\mathbf{g}_2$ monomers can replace the $\mathbf{u}_{2,j}$ monomers.

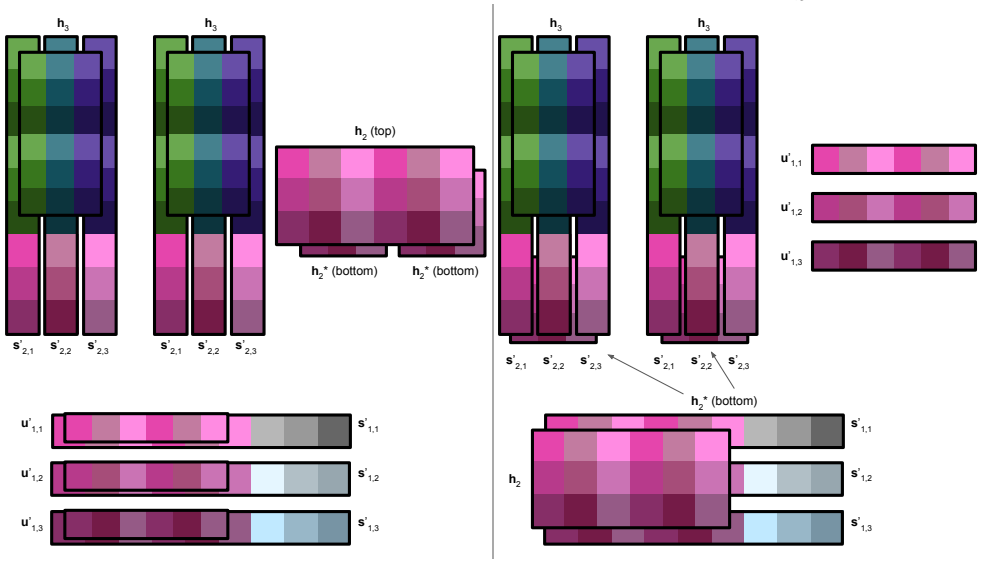


FIGURE 2.7. The converging translator gadget, before and after it is triggered to propagate the signal forward by one layer. When two copies each of $\mathbf{s}'_{2,1}$, $\mathbf{s}'_{2,2}$ and $\mathbf{s}'_{2,3}$ have been brought together into two complexes, instead of directly replacing all the $\mathbf{u}'_{1,j}$ monomers, they can split a $\{\mathbf{h}_2, \mathbf{h}_2^*, \mathbf{h}_2^*\}$ complex, and the $\mathbf{h}_2$ monomer can replace the $\mathbf{u}'_{1,j}$ monomers. Note that the only way to propagate the signal efficiently is to use the translator gadget; not using it requires 3 splits and 4 merges, showing that this TBN no longer has an entropy gap. If instead we hadn't used the translator gadgets in the previous layer, then all six $\mathbf{s}'_{2,j}$ monomers in this image would be together in a single complex, and it would be equally efficient to either use this translator gadget or directly displace the $\mathbf{u}'_{1,j}$.

(1) $\widetilde{T}$ *has exactly one stable configuration* $\widetilde{\sigma}_{n,k}$*, and* $d(\widetilde{T}, \widetilde{T}^{\text{a}}) > 2^n$*.*

(2) $\widetilde{T}$ *has an entropy gap of* $\frac{k}{2}$*, and* $\widetilde{T}^{\text{a}}$ *has the property that all of its configurations* α *that are within distance to stability* $\frac{k}{2}$ *satisfy* $d(\widetilde{\sigma}_{n,k}, \alpha) > 2^n$*.*

(3) $\widetilde{T} = \widetilde{T}_{n,k}$ *uses* $O(nk)$ *total monomer types,* $O(nk^2)$ *domain types, and* $O(k^2)$ *domains per monomer.*

(4) The unique stable configuration of $\widetilde{T}$ *has* $O(k)$ *monomers in its largest polymer. There is a stable configuration of* $\widetilde{T}^{\text{a}}$ *sharing this property.*

Compared to Theorem 2.3.1, this theorem trades away the condition that both TBNs have only a single stable configuration in exchange for the post-analyte TBN having a configuration with $O(k)$ monomers per polymer, whereas the previous construction has roughly $k \cdot 2^n$ monomers in a single polymer.

The second condition is somewhat complex. This complexity's necessity is explained by Fig. 2.7. In that figure, if we propagate signal without using the translator gadget, we arrive at a configuration that is saturated but has only one fewer complex than a stable configuration. However, such near-stable configurations are still very different from the stable configuration of $\widetilde{T}_{n,k}$, so it is still possible to distinguish the two TBNs with an amplification factor proportional to 2^n and a resilience to false positives and negatives proportional to k.

Proof. Recall the constructions of $\widetilde{T}_{n,k}$ and $\widetilde{T}^{\text{a}}_{n,k}$ from Section 2.3.3. Our argument will be very similar to that of Theorem 2.3.1 (i.e., the above lemmas), except we need to account for the extra monomer types.

First, consider $\widetilde{T}_{n,k}$, where **a** is absent. We wish to show that its stable configuration looks like that of $T_{n,k}$, with the added **g** and **h** monomers only binding to added $\mathbf{g}^*$ and $\mathbf{h}^*$ monomers respectively. We order the merges to get to a saturated configuration in essentially the same order as we did in analyzing $T_{n,k}$: first we will make all merges necessary to cover all $(1, j, \ell)^*$ sites, then $(2, j, \ell)^*$, and so on up to $(n+1, j, \ell)^*$, then $(n, j, \ell)'^*$, and so on. As before, at each step, we will see that we cannot make merges in any way other than those in the desired stable configuration without needing $k-1$ extra merges for that step.

For $(i, j, \ell)^*$ sites, at each step, there is exactly one way to cover all starred sites by making one merge per monomer with these starred sites: we cover each $\mathbf{s}_{i,j}$ with a $\mathbf{u}_{i,j}$ and each $\mathbf{g}^*_i$ with a $\mathbf{g}_i$.

In particular, we already know from the proof of Lemma 2.3.2 that this is true for the $\mathbf{s}_{i,j}$ if we only use $\mathbf{s}_{i-1,j}$ and $\mathbf{u}_{i,j}$ to cover it, and that we will otherwise need to make $k-1$ extra merges. Clearly we also cannot cover $\mathbf{g}_i^*$ with anything other than $\mathbf{g}_i$ without making k merges to cover it (and thus $k-1$ extra merges), so we cannot use $\mathbf{g}_i$ to cover $\mathbf{s}_{i,j}$.

Likewise, for $(i,j,\ell)'^*$ sites, we only need to observe that each $\mathbf{h}_i^*$ monomer can only be covered in a single merge by $\mathbf{h}_i$, so any other way of making merges necessarily involves $k-1$ extra merges. So by the same argument as in Lemma 2.3.2 and Corollary 2.3.3, $\widetilde{T}_{n,k}$ has exactly one stable configuration with an entropy gap of $\frac{k}{2}$. This configuration has $1+\frac{k}{2}$ monomers in the polymer containing $\mathbf{p}^*$ and all the $\mathbf{p}_j$, 3 monomers in each $\{\mathbf{h}_i, \mathbf{h}_i^*, \mathbf{h}_i^*\}$ polymer, and 2 monomers in each other polymer.

Now, consider $\widetilde{T}^{\mathbf{a}}_{n,k}$, where $\mathbf{a}$ is present. If we take the stable configuration of $T^{\mathbf{a}}_{n,k}$ and simply put all $\mathbf{g}$ monomers into $\{\mathbf{g}_i, \mathbf{g}_i^*\}$ polymers, and all the $\mathbf{h}$ monomers into $\{\mathbf{h}_i, \mathbf{h}_i^*, \mathbf{h}_i^*\}$ polymers, we have still made exactly one merge per monomer with any starred sites, so by Corollary 2.2.12 it is stable. If we then carry out the shifts described in Fig. 2.6 and Fig. 2.7, an equal number of merges and splits are made at each step, so the resulting saturated configuration is still stable. Additionally, in this configuration, the largest polymers have $k+3$ monomers (specifically, those containing a set of $\mathbf{s}_{i,j}$ along with one copy of $\mathbf{g}_i$ and two copies of $\mathbf{g}_{i+1}^*$).

All that remains to show is that all configurations of $\widetilde{T}^{\mathbf{a}}_{n,k}$ that are within $\frac{k}{2}$ distance to stability have exponentially many different polymers from the stable configuration of $\widetilde{T}_{n,k}$. We will do this by showing that all $\mathbf{u}$ and $\mathbf{u}'$ monomers are free in all such configurations.

Again, this argument is very similar to the argument without the translator gadgets in Lemma 2.3.4. We consider merges to cover starred sites in the opposite order of the above argument for $\widetilde{T}_{n,k}$. First, consider the merges necessary to cover all the $(1,j,\ell)'$ starred sites (on $\mathbf{p}^*$). Like before, they must be covered by either all the $\mathbf{u}'_{1,j}$ monomers or all the $\mathbf{p}_j$ monomers, but using the latter gives a feed-forward configuration in which $\frac{k}{2}-1$ extra merges have already been made. Thus, to be within $\frac{k}{2}$ distance to stability, we must use the $\mathbf{s}'_{1,j}$. Next, for the $(2,j,\ell)'$ starred sites, with the merges already made, there are two copies of each of these sites all together on the polymer containing all the $\mathbf{s}'_{1,j}$, and one copy of each site on each of the two $\mathbf{h}_2^*$ monomers. If we are to merge any $\mathbf{u}'_{1,j}$ monomers to any of these in such a way that they cannot be split off without the result still being saturated, then we must merge all of the $\mathbf{u}'_{1,j}$ into one polymer. Like with the

argument for $\widetilde{T}^{\mathrm{a}}_{n,k}$, we see that this results in $k-1$ extra merges compared to a stable configuration. Thus, we cannot use any $\mathbf{u}'_{1,j}$, and these sites must be covered by the $\mathbf{s}'_{2,j}$ and $\mathbf{h}_2$ monomers.

We may do this either by using the $\mathbf{h}_2$ to cover both $\mathbf{h}^*_2$ (in effect, not using the translator gadget) or by using $\mathbf{h}_2$ to cover all the $\mathbf{s}'_{1,j}$. The only difference in terms of the argument is that in the former case *all* of the $\mathbf{s}_{2,j}$ will be brought together in a single polymer, and in the latter case they will be split between two polymers. In the former case, it may require one extra merge to use translator gates in the next layer; however, either way, the same argument on each other layer in sequence shows that we cannot use any $\mathbf{u}'_{i,j}$ monomers without suffering $\frac{k}{2}-1$ extra merges. Likewise, the exact same argument shows that the same thing is true of $\mathbf{u}_{i,j}$ monomers, necessitating that in any configuration that makes fewer than $\frac{k}{2}-1$ extraneous merges, all exponentially many $\mathbf{u}$ and $\mathbf{u}'$ monomers must be free, as desired. □

2.4. Upper Limit on TBN Signal Amplification

In this section, we show the following theorem providing an upper bound on the distance between a TBN before and after adding a single copy of a monomer, showing that the distance is at most double-exponential in the "size" of the system:

THEOREM 2.4.1. *Let T be a TBN with d domain types, m monomer types, and at most a domains on each monomer. Let $n = \max\{d, m, a\}$. Let T' be T with one extra copy of some monomer. Then $d(T, T') \leq n^{8n^{7n^2}}$.*

Recall Definition 2.2.7 for the distance between TBNs. Essentially, this theorem is saying that adding a single copy of some monomer can only impact doubly exponentially many total polymers, no matter how many total copies of each monomer are in the TBN.

Our strategy for proving this theorem is to fix some ordering on polymer types, and bound the distance between the lexicographically earliest stable configuration of an arbitrary TBN under that ordering before and after adding a single copy of some monomer. To bound this distance, we cast the problem of finding stable configurations of a TBN as an integer program (IP), and use methods from the theory of integer programming value functions to give a bound on how much the solution to this IP can change given a small change in the underlying TBN.

We first introduce a definition from [34] and some notation that was unnecessary in previous sections.

Definition 2.4.2. *Given a (star-limiting) TBN T, the* polymer basis *of T, denoted $\mathcal{B}(T)$, is the set of polymers $\mathbf{P}$ such that both of the following hold:*

- $\mathbf{P}$ *appears in some saturated configuration of a star-limiting TBN using the same monomer types as T.*
- $\mathbf{P}$ *cannot be split into two or more self-saturated polymers.*

The polymer basis is a useful construction because it is known to describe exactly those polymer types that may appear in stable configurations of T. It is always finite, and we will bound its size later.

Given a TBN T, let $M(T)$ denote its monomer types, and let $T(\mathbf{m})$ denote the count of monomer $\mathbf{m}$ in T. Given a polymer $\mathbf{P}$ and a monomer type $\mathbf{m} \in M(T)$, let $\mathbf{P}(\mathbf{m})$ represent the count of monomer $\mathbf{m}$ in polymer $\mathbf{P}$.

Suppose for the rest of this section that we have a TBN T to which we wish to add a single copy of some monomer $\mathbf{a}$ (which may or may not exist in T). Let T' be T with $\mathbf{a}$ added.

2.4.1. Finding Stable Configurations via Integer Programming. Prior work [34] has shown that the problem of finding the stable configurations of a TBN can be cast as an IP. There are multiple different formulations; we will use a formulation that is better for the purpose of reasoning theoretically about TBN behavior.

Let $\{x_{\mathbf{P}} : \mathbf{P} \in \mathcal{B}(T)\}$ be variables each representing the count of polymer $\mathbf{P}$ in a configuration of T. Then consider the following integer programming problem:

$$\begin{aligned} \max \quad & \sum_{\mathbf{P} \in \mathcal{B}(T)} x_{\mathbf{P}} \\ \text{s.t.} \quad & \sum_{\mathbf{P} \in \mathcal{B}(T)} \mathbf{P}(\mathbf{m}) x_{\mathbf{P}} = T(\mathbf{m}) \quad \forall \mathbf{m} \in M(T) \\ & x_{\mathbf{P}} \in \mathbb{N} \quad \forall \mathbf{P} \in \mathcal{B}(T) \end{aligned} \tag{2}$$

Intuitively, the linear equality constraints above express "monomer conservation": the total count of each monomer in T should equal the total number of times it appears among all polymers.

The following was shown in [34]; for the sake of self-containment, we show it here as well

Proposition 2.4.3. *The optimal solutions to the IP (2) correspond exactly to stable configurations of T.*

Proof. If the variables $x_{\mathbf{P}}$ form a feasible solution, then those counts of polymers are a valid configuration because they exactly use up all monomers. If the solution is optimal, then there is no saturated configuration with more polymers (as only polymers from $\mathcal{B}(T)$ can show up in stable configurations), so the configuration is stable. Conversely, if a configuration σ is stable then it can be translated into a feasible solution to the IP because it only uses polymers from $\mathcal{B}(T)$ and obeys monomer conservation. If there were a solution with a greater objective function, then this would translate to a configuration with more complexes that is still saturated (because all polymers in the polymer basis are self-saturated), contradicting the assumption of σ's stability. □

We observe that adding an extra copy of some monomer to a TBN corresponds to changing the right-hand side of one of the constraints of this IP by one. Note that this is true even if we add a copy of some monomer for which there were 0 copies, as we may still include variables for polymers that contain that monomer in the former IP and simply consider there to be 0 copies of the monomer. Therefore, we are interested in sensitivity analysis of how quickly a solution to an IP can change as the right-hand sides of constraints change.

However, there is one edge case we must account for first. It is possible that T and T' have different polymer bases. This is because of the first requirement in Definition 2.4.2 requiring that the polymer basis respects that starred sites are limiting. If we add a single copy of a monomer, this may change which sites are limiting, if $\mathbf{a}$ has more copies of a starred site than T had excess copies of the unstarred site. We cannot include variables for such polymers in the IP formulation without taking extra precautions, as if we do there may be optimal solutions that don't correspond to saturated configurations. Therefore, we will first account for how many copies of such a polymer T and T' may differ by:

Lemma 2.4.4. *Suppose that some polymer $\mathbf{P}$ is exactly one of $\mathcal{B}(T')$ and $\mathcal{B}(T)$. Then any saturated configuration of T' contains at most $|\mathbf{a}|$ copies of $\mathbf{P}$, where $|\mathbf{a}|$ denotes the number of sites on $\mathbf{a}$.*

Note that this result is slightly surprising—one natural way that one might try to design a TBN that amplifies signal is by designing the analyte so that it intentionally flips which sites are limiting. This result shows that this is an ineffective strategy: going from 5 excess copies of some

site a to 5 excess copies of a^* is seemingly no more helpful in instigating a large change than going from 60 excess copies of a to 50.

PROOF. If $\mathbf{P}$ is in $\mathcal{B}(T')$ but not $\mathcal{B}(T)$, it must contain an excess of a starred site that was limiting in T, but is no longer limiting in T'. We see this because $\mathbf{P}$ necessarily occurs in a saturated configuration of the TBN containing precisely the monomers that it is composed of; therefore, in order to not be in $\mathcal{B}(T)$, by definition of the polymer basis, it must be the case that this TBN has different limiting sites than T'.

Let a denote some such site type, so that a^* is limiting in T and a is limiting in T', and $\mathbf{P}$ contains an excess of a^*. Then $\mathbf{a}$ must contain an excess of a^*, but it cannot contain more than $|\mathbf{a}|$ excess copies. Therefore, there are at most this many *total* excess copies of a^* in T'. It follows that if there are more than $|\mathbf{a}|$ copies of $\mathbf{P}$ in a configuration of T', then those copies of $\mathbf{P}$ collectively have more excess copies of a^* than T' does, so some other polymer in that configuration would have to have an excess of a. This implies that such a configuration is not saturated (and therefore also cannot be stable). An identical argument shows that the same is true for polymers in $\mathcal{B}(T)$ but not $\mathcal{B}(T')$. □

In order to analyze and compare the two IP instances, we need them to have the same variable set. Therefore, we will include variables for all polymers from both polymer bases in both IP formulations. Let $\mathcal{B}(T, T') = \mathcal{B}(T) \cup \mathcal{B}(T')$ denote this merged polymer basis, and let $P = |\mathcal{B}(T, T')|$ denote the total number of possible polymers we must consider, or equivalently the number of variables we will have in these IPs. In each IP, we will have a constraint on each variable representing a polymer not in the relevant polymer basis, that says that that variable must equal zero.

2.4.2. Sensitivity Analysis. This sensitivity analysis problem of how IPs change as the right-hand sides of constraints change was studied by Blair and Jeroslow in [9]. We will not need their full theory, but we will use some of their results and methods.

In Corollary 4.7 of [9], they show that there is a constant K_3, independent of the right-hand sides of constraints (in our case, independent of how many copies of each monomer exist) such that:

$$R_c(v) \leq G_c(v) \leq R_c(v) + K_3, \tag{3}$$

where $G_c(v)$ gives the optimal value of the objective function c of a minimization IP as a function of the vector v of right-hand sides of constraints, and $R_c(v)$ gives the optimal value of the same problem when relaxing the constraint that variables must have integer values. The objective function we've shown so far is to maximize the sum of polymer counts rather than minimize, but the same statement applies that the integer and real-valued optimal solutions differ by at most K_3. In defining K_3, they also show the existence of a constant M_1 such that

$$|R_c(v) - R_c(w)| \leq M_1 \, ||v - w|| \,, \tag{4}$$

where v and w are different vectors for the right-hand sides of constraints. Note that we take all norms as 1-norms. Combining these inequalities, we see that

$$|G_c(v) - G_c(w)| \leq M_1 \, ||v - w|| + K_3. \tag{5}$$

For example, if we want to know the difference between the total number of polymers in a stable configuration before and after adding one copy of a monomer (and if the polymer bases of T and T' are identical), then we care about increasing one element of v by 1, so our bound on this difference is $M_1 + K_3$. This statement applies to maximization and minimization problems.

2.4.3. From Optimal Values to Polymer Counts. For ease of analysis, we order the polymers in $\mathcal{B}(T, T')$ as follows: first we list all the polymers that are not in $\mathcal{B}(T)$, then all the polymers that are not in $\mathcal{B}(T')$, then all the polymers in $\mathcal{B}(T) \cap \mathcal{B}(T')$. We need to show that the number of copies of each individual polymer does not change too much. We do this using a technique similar to Corollary 5.10 in [9].

Let P_{tot} be the total number of polymers in a stable configuration (either before or after adding $\mathbf{a}$, depending on which case we are examining).

We now define a new sequence of integer programs whose optimal values give polymer counts in the lexicographically earliest stable configuration under this ordering. We do this by finding the value of each variable $x_{\mathbf{P}}$ in order. This sequence of IP problems is defined separately for both TBNs, before and after adding $\mathbf{a}$.

For those variables representing a polymer that is in one basis but not the other, we do not need to analyze this IP, so we simply fix such a variable's value to whatever its value is in this

lexicographically earliest stable configuration, which will be 0 in one TBN and bounded by $|\mathbf{a}|$ by Lemma 2.4.4 in the other.

Now, to find the value of some particular variable $x_{\mathbf{Q}}$ in either of the two TBNs where $\mathbf{Q} \in \mathcal{B}(T) \cap \mathcal{B}(T')$, suppose we have already found the value $y_{\mathbf{P}}$ we wish to fix $x_{\mathbf{P}}$ to for each $\mathbf{P} < \mathbf{Q}$ under our ordering. Then we define a new IP on all the same variables as follows:

$$
\begin{aligned}
\min \quad & x_{\mathbf{Q}} \\
\text{s.t.} \quad & \sum_{\mathbf{P} \in \mathcal{B}(T,T')} x_{\mathbf{P}} = P_{tot} \\
& \sum_{\mathbf{P} \in \mathcal{B}(T,T')} \mathbf{P}(\mathbf{m}) x_{\mathbf{P}} = T(\mathbf{m}) \quad \forall \mathbf{m} \in m(T) \quad 3 \\
& x_{\mathbf{P}} = y_{\mathbf{P}} \quad \forall \mathbf{P} < \mathbf{Q} \\
& x_{\mathbf{P}} \in \mathbb{N} \quad \forall \mathbf{P} \in \mathcal{B}(T, T')
\end{aligned} \tag{6}
$$

By construction, this IP gives us the smallest possible value that $x_{\mathbf{Q}}$ can take on in a stable configuration (as all variables must sum to P_{tot}) in which all previous $x_{\mathbf{P}}$ have fixed values. Then this process gives us a sequence of P (minus however many polymers were only in one polymer basis) different pairs of IP problems that we can sequentially compare to bound the differences between the values of the individual polymer counts in these lexicographically earliest configurations. We can repeatedly apply Eq. (5) to each $x_{\mathbf{P}}$ in turn, as each variable's value before and after adding $\mathbf{a}$ will be given by the optimal value of (6) where the only differences are in the right-hand sides of constraints. The remaining proof of Theorem 2.4.1 consists mostly of making these bounds concrete.

PROOF OF THEOREM 2.4.1. Recall we are trying to sequentially bound the difference between the amounts $x_{\mathbf{P}}$ of polymers in the lexicographically earliest configurations of T and T', an arbitrary TBN before and after adding a copy of a monomer $\mathbf{a}$. We defined a sequence of IPs in Eq. (6) whose optimal values give the amounts of polymers in these configurations. Throughout we will be loose with coefficients as our main concern is showing that the bound is doubly exponential.

For those $x_{\mathbf{P}}$ representing polymers in only one polymer basis, we already know that their difference is bounded by $|\mathbf{a}|$. To be conservative, we will both assume that we must account for

this difference for all P variables, and then carry out the rest of the analysis as though we must account for all P variables being in $\mathcal{B}(T) \cap \mathcal{B}(T')$ (as these latter variables will actually contribute more to the analysis).

Under these assumptions, we can bound the difference between P_{tot} (the total number of polymers in a stable configuration) for the two TBNs: P_{tot} is the optimal value of the IP in Eq. (2) when adding in constraints that all these variables that are only in one of the two polymer bases have specific values (which are either 0, or bounded by $|\mathbf{a}|$). Eq. (2) differs between the two cases by these P constraints differing by up to $|a|$ and one additional constraint $T(\mathbf{a})$ differing by 1 because of the one extra copy of $\mathbf{a}$. Therefore, by Eq. (5), P_{tot} differs between the two by at most $M_1(P|\mathbf{a}|+1)+K_3$.

Now, we account for all variables that are in $\mathcal{B}(T) \cap \mathcal{B}(T')$. For the first such $x_{\mathbf{P}}$, we see that P_{tot} may have changed by $M_1(P|\mathbf{a}|+1)+K_3$, $T(\mathbf{a})$ has changed by a fixed 1, and up to P of the variables representing polymers in only one polymer basis may have been fixed in value. Thus, we can bound the difference between the norm of the right-hand side constraint vectors in the two versions of Eq. (6) for this first such variable by:

$$P|\mathbf{a}| + M_1(P|\mathbf{a}|+1) + K_3 + 1 \leq 2M_1P|\mathbf{a}| + K_3. \tag{7}$$

It follows by Eq. (5) that the difference between the value of this first $x_{\mathbf{P}}$ before and after adding one copy of a monomer is bounded by $M_1(2M_1P|\mathbf{a}|+K_3)+K_3$. Then for the next variable in order, the value of $x_{\mathbf{P}}$ is baked in as the right-hand side of a constraint, meaning that this difference now contributes to the value $||v-w||$ in Eq. (5). Thus, if $\mathbf{P}_i$ denotes the ith polymer in our ordering, we obtain a recurrence relation yielding a bound B_i on the difference between $x_{\mathbf{P}_i}$ before and after adding $\mathbf{a}$, for $1 \leq i \leq P$ (where B_0 is defined as a base case):

$$\begin{aligned} B_0 &= 2M_1P|\mathbf{a}| + K_3 \\ B_i &= K_3 + M_1 \sum_{j=0}^{i-1} B_j \end{aligned} \tag{8}$$

We can bound $\sum_{j=0}^{i-1} B_j$ by $2B_{i-1}$ as this sequence clearly grows faster than 2^i, which allows us to solve the recurrence to see that all terms of B_i are subleading to $K_3 \cdot (2M_1)^i$ and each term can be safely bounded by $2K_3 \cdot (2M_1)^i$.

Thus, the total distance between these stable configurations is bounded by the sum of P terms that each have this bound when we replace i with P, giving a bound of $2PK_3 \cdot (2M_1)^P$.

We now bound these individual values. Let S be the maximum number of monomers in any polymer. The value K_3 in [9] is constructed from three other values as the expression $M_1M_3 + M_2$. These values can each be bounded, some based on another constant K_2 which will be bounded next:

(1) M_1 is essentially a bound on how quickly the objective value of the corresponding real-valued linear program can change as the right-hand side changes, as described earlier. The optimal value of a linear program is always one of some set of $\lambda_i \cdot v$ where the λ_i vectors come from the extreme points of a polyhedron based on the IP; [9] bounds M_1 by the maximum norm of these λ_i. A bound in our case is P: if there was a λ_i whose elements summed to more than P, then this would imply we could get a configuration with more total polymers (as the sum of polymer counts is our objective function) than monomers, which is impossible.

(2) M_2 is the maximum value of the objective function when all variables are at most K_2, which in this case is PK_2.

(3) M_3 is the maximum norm of the constraint matrix times the vector of variables when all variables are at most K_2. In TBN language, this gives the number of monomers present in a configuration with K_2 copies of every polymer in the polymer basis. The elements of the constraint matrix are bounded by S, so M_3 is bounded by SPK_2.

Thus, $K_3 = M_1M_3 + M_2 \leq 2SP^2K_2$. The value of K_2 is constructed by taking subsets B of variables with the following property: the space of a_j such that $\sum a_j x_j = 0$ is one-dimensional. In TBN language, an element of this subspace corresponds to a pair of configurations α and β of some TBN such that the polymer types present in α and the polymer types present in β are disjoint subsets of B. In other words, it represents a way to take a configuration using polymer types in B and reconfigure it to use entirely different polymer types in B. This can be seen by letting positive a_j give counts of polymers in α and negative a_j give counts of polymers in β. The value K_2 is as large as the greatest number of a single polymer type that may be necessary for such a reconfiguration.

Then to bound this, we observe that we can find the solutions to this homogeneous system of equations by doing Gaussian elimination on an at most $n \times P$ (as there are at most n monomer types) matrix whose elements are bounded by S. A simple bound for the largest element that can occur in this Gaussian elimination is S^{n+1}, so this is also a bound on our constant K_2.

Thus, $K_3 \leq 2S^{n+2}P^2$.

Finally, we bound S and P. [23] shows that for a TBN with d domain types, m monomer types and a domains per monomer, the largest polymer in any stable configuration has size at most $2(m+d)(ad)^{2d+3}$. Since n is a uniform bound on all these values, the maximum size of any polymer is $4n \cdot (n^{4n+6}) \leq n^{5n}$. That is to say, $S \leq n^{5n}$. Since there are at most n different monomer types, the number of polymers of a given size i that can be formed out of them is at most i multichoose $n = \binom{i+n-1}{n} \leq (i+n-1)^n$. Therefore, the total number of possible polymer types in the polymer basis is bounded by:

$$\begin{aligned} P &\leq \sum_{i=1}^{S} (i+n-1)^n \\ &\leq S \cdot (S+n-1)^n \\ &\leq n^{5n}(n^{5n}+n-1)^n \\ &\leq n^{6n^2}. \end{aligned} \tag{9}$$

Thus, we finally obtain the following bound on the distance between these configurations σ and σ':

$$\begin{aligned} d(\sigma,\sigma') &\leq 2PK_3 \cdot (2M_1)^P \\ &\leq 4S^{n+2}P^3 \cdot (2P)^P \\ &\leq n^{6n^2}P^3 \cdot (2P)^P \\ &\leq n^{24n^2} \cdot (2n^{6n^2})^{n^{6n^2}} \\ &\leq n^{24n^2} \cdot (n^{7n^{7n^2}}) \\ &\leq n^{8n^{7n^2}}. \end{aligned} \tag{10}$$

□

2.5. Conclusion

In this chapter we have defined the signal amplification problem for Thermodynamic Binding Networks, and we have demonstrated a TBN that achieves exponential signal amplification. We also showed a doubly-exponential upper bound for the problem. As TBNs model mixtures of DNA, a TBN that amplifies signal can potentially be implemented as a real system. An upper bound has implications for how effective a system designed in this way can potentially be, and shows that there are some limitations for a purely thermodynamic approach to signal detection and amplification.

One future goal would be to bridge the gap between our singly exponential amplifier and doubly exponential upper bound by either describing a TBN that can amplify signal more than exponentially, or deriving a more precise upper bound. If one wished to construct a TBN with doubly exponential amplification, an examination of our upper bound proof will show that such a TBN must have an exponentially sized polymer basis, and most likely would need to actually use an exponential amount of different polymer types in its stable configurations either with or without the analyte. Such a design seems relatively unlikely to come to fruition, and it seems more likely that our proof technique or similar techniques can be tightened in order to show a stricter upper bound. Thus, we conjecture that the true upper bound is (singly) exponential.

There are also other types of robustness that we have not discussed in this work that merit further analysis. One of these is input specificity: the question of how well the system amplifies signal if the analyte is changed slightly. Another is sensitivity to the number of copies of each component. Intuitively, our system's behavior depends on having exactly equal numbers of complementary strands within each layer; if there are too many copies of one, it may result in those excess copies spuriously propagating or blocking signal to the next layer. This issue may be intrinsic to thermodynamic signal amplifiers, or there may be some system more robust to it. Lastly, it may be experimentally useful to show that our system achieves its stable states not only in the limit of thermodynamic equilibrium, but also more practically when annealed. Some systems such as HCR are designed to reach non-equilibrium, meta-stable states when annealed. We conjecture that our system should reach equilibrium when annealed, because kinetic traps in the system are far away from being thermodynamically stable (large entropy gap). Formally studying annealing

could be done by analyzing versions of the TBN model with different tradeoffs between entropy and enthalpy to model different temperatures.

CHAPTER 3

Thermodynamic Signal Amplifier Experiments

The work presented in this chapter was done in David Soloveichik's lab at the University of Texas in Austin, with help from him and his students and postdocs, especially Boya Wang and Tony Szlegowski.

3.1. Introduction

In order to verify the theory described in the previous chapter, we designed real DNA strands that implement the binding logic of our TBN. We carried out experiments to demonstrate that the system can amplify signal purely thermodynamically. All experiments were carried out by mixing together strands, annealing once, and then examining the result to see what strands were binding, primarily through the use of PAGE gels.

3.2. Single-stranded motif

Our first round of experiments were conducted using a motif where each TBN monomer corresponds to a single strand of DNA, as in Fig. 3.1. Each domain has 10 or 11 bases, alternating in such a way to ensure that crossovers are aligned as seen in Fig. 3.2. This serves as the most direct realization of the semantics of TBNs. Two strands sharing a sufficiently long complementary domain will generally bind at room temperature, and strands without complementary domains generally will not interact. To ensure this latter feature, we used the nuad package [2] to design DNA sequences that avoided too much undesired interaction. nuad uses NUPACK [27] and ViennaRNA [40] to evaluate the minimum free energy of strands or of *unpseudoknotted* secondary structures among multiple strands. Psuedoknottedness is a technical condition describing DNA binding patterns, typically of interest because it is generally NP-hard to evaluate the minimum free energies of complexes when pseudoknotted secondary structures are allowed to form. Because most of the structures we expect to form are pseudoknotted, we therefore cannot evaluate their free energies easily. Thus, our approach was to design sequences such that individual strands and

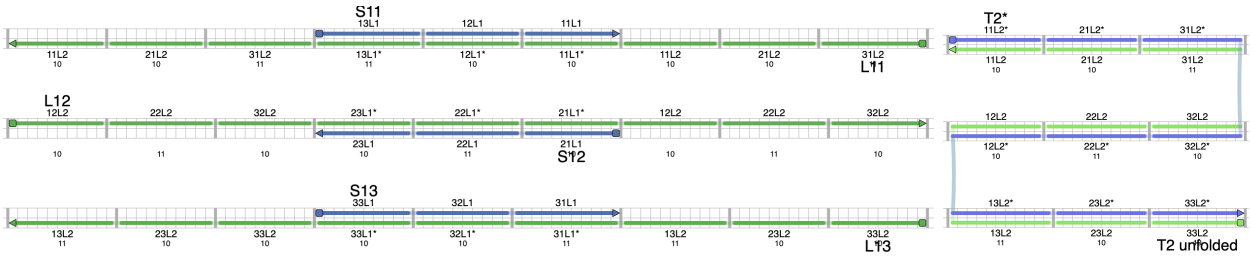


FIGURE 3.1. The first motif used for signal amplification experiments, shown using scadnano [22]. Left: strands corresponding to **s** and **u** monomers. Right: strands corresponding to **g** and **g*** monomers.

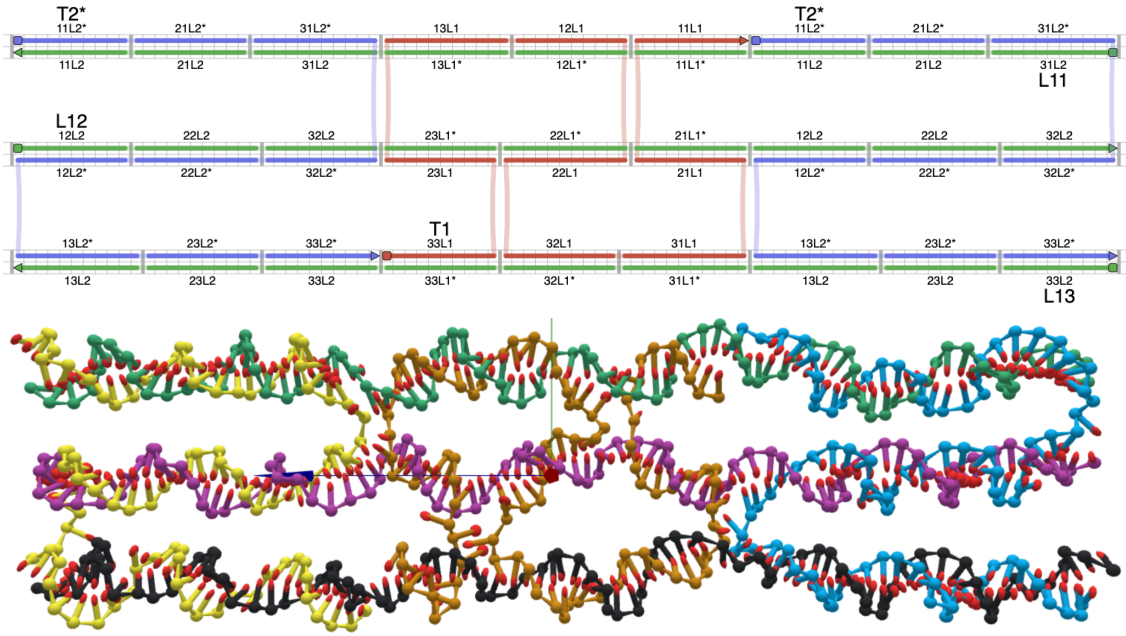


FIGURE 3.2. The most complicated polymer expected to form. Top: view of strands at domain level in scadnano. Bottom: rendered view of expected binding pattern in OxDNA [63].

complexes of a single pair of strands (which we do not expect to bind in a pseudoknotted way) did not have too low of free energies.

We ran experiments by mixing strands together, annealing overnight from 90-20°C. We then ran our mixes in pre-cast PAGE gels. PAGE gels were chosen because they allowed us to distinguish small strands and complexes, with our smallest strands being 31 bases long. The results of our last experiment with this motif can be seen in Fig. 3.3. While this experiment does not show amplification, it shows that it is possible for signal to pass between layers of our system in the way we expect it to. We encountered many issues that seemed to stem from relatively complex

polymers not forming when we expected them to based on the TBN model. We conjectured that this is because these complexes were highly geometrically constrained, giving them entropic penalties not captured in the TBN model.

3.3. Scaffolded motif

In order to fix the perceived geometric issues with the motif we used for our first design, we built a new design starting over from the semantics of the amplification TBN. Our second round of experiments consisted of similar experiments, conducted with a set of strands that implement the amplifier's logic in a different way. In this "single-stranded scaffold" motif, a single TBN monomer is reperesented by one scaffold strand, plus one handle strand for each domain. Each handle strand has a unique domain in common with the corresponding scaffold strand, ensuring that these strands should always be found together at sufficiently low temperatures when handles are placed in excess of scaffolds.

While we conducted similar experiments as the first motif, we were not able to find conclusive evidence that the system was working as expected. The primary issue we encountered was that many strands generally did not seem to bind to each other at room temperature when we expected them to do so. Most complexes that we expected to form involve monomers binding along at least three domains. Each handle strand's binding domain for complementary handle strands contained only 7 bases. While we expected this to be sufficient when multiple handles were all binding together, it is possible that there were unexpected geometric or kinetic barriers to the formation of the complexes we expected to form. It is also possible that this motif modifies the thermodynamics of the system in such a way that it does not match the theoretical predictions of the TBN model as well. Further research is warranted to find if this motif is suitable, or if others exist that might better be able to match the model.

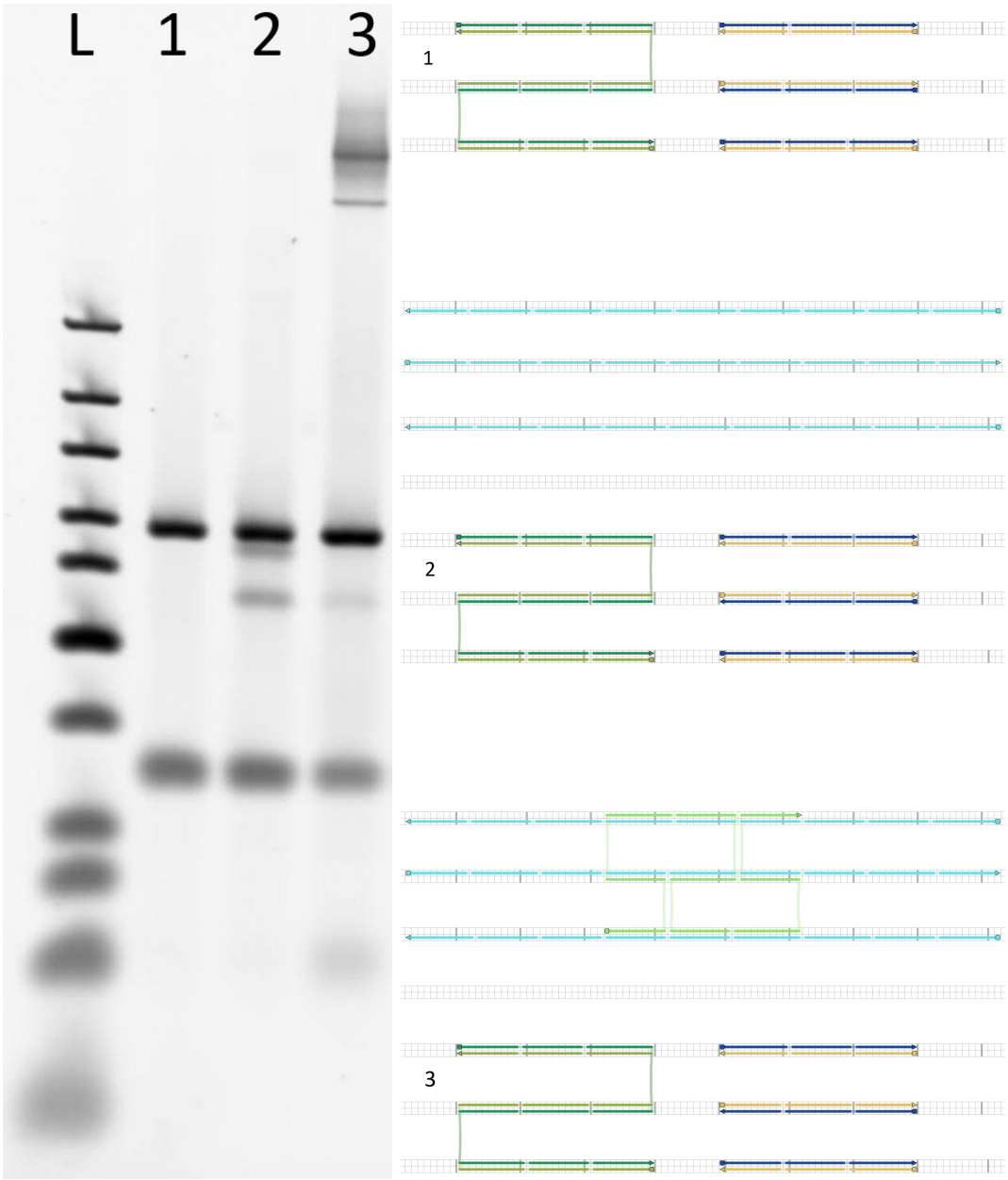


FIGURE 3.3. The last experiment conducted using the motif described in Section 3.2. The lane marked 'L' contains ThermoFisher Ultra Low Range DNA Ladder. The faint band near the bottom of lane 3, absent from the other lanes, contains amplifier strands that have been separated from their complementary strands (dark blue and yellow, lower-right). These strands have no domains in common with the previous layer (light blue/light green, top). This shows that the analyte from the previous layer (light green) can act on the next layer in a way that is mediated through the translator gate (dark green and yellow, lower left).

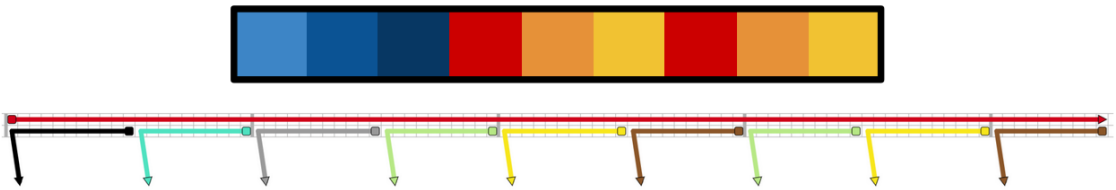


FIGURE 3.4. The single-stranded scaffold motif. The monomer with 9 domains is represented by one scaffold strand, together with 9 handle strands that bind to it.

CHAPTER 4

Fast Exact Simulation of Stochastic Chemical Reaction Networks

This chapter is joint work with David Doty. It and most of the following chapter were originally published as [46].

4.1. Introduction

The model of *chemical reaction networks* (*CRNs*) is one of the oldest and most widely used in natural science [33]. It describes reactions between abstract species, for example $A + B \xrightarrow{k} C + 2D$, which indicates that if a molecule of type A (a.k.a., *species* A) collides with a molecule of type B (A and B are the *reactants*), they may split into three molecules, one of type C and two of type D (the *products*). The number $k \in \mathbb{R}_{>0}$ is called a *rate constant*. The standard algorithm for stochastically simulating CRNs is the *Gillespie algorithm* [30], which we now describe.

The Gillespie algorithm iteratively executes one reaction at a time, taking time $\Theta(\ell)$ for ℓ reactions. A *configuration* is a multiset of species; equivalently a nonnegative integer vector $\mathbf{c}$, where $\mathbf{c}(S) \in \mathbb{N}$ denotes the *count* of S; when $\mathbf{c}$ is clear from context, we write $\#S$ to denote $\mathbf{c}(S)$. A reaction is more likely to occur the more of its reactants there are; more precisely, given a fixed *volume* $v \in \mathbb{R}_{>0}$, the *propensity* (a.k.a., *rate*) of a reaction is proportional to the product of reactant counts and rate constant k, divided by v^{r-1}, where r is the number of reactants. For example the propensity of the reaction $A + B \xrightarrow{k} C + 2D$ is $\frac{k \cdot \#A \cdot \#B}{v}$. (See Section 4.2.1 for a detailed definition.) The Gillespie algorithm repeatedly samples a reaction to execute (for example executing the above reaction would decrease $\#A, \#B$ by 1, increase $\#C$ by 1, and increase $\#D$ by 2), with probability proportional to its propensity. The time until this occurs is an exponential random variable with rate equal to the sum of all reaction propensities. In other words the model is a continuous-time Markov chain whose states are CRN configurations, with transitions determined by reactions.

A distributed computing model known as *population protocols* [6], originally defined to model mobile finite-state sensor networks, turns out to be equivalent to the special case of CRNs in which every reaction has exactly two reactants, two products, and rate constant $k = 1$. In the population

protocol model, a scheduler repeatedly picks a pair of distinct molecules A, B uniformly at random from a population of n molecules and has them *interact*, which means replacing them with C and D if there is a reaction $A+B \to C+D$, or doing nothing if there is no such reaction (i.e., simulating the "null" reaction $A + B \to A + B$). In the natural timescale in which we expect each molecule to have $O(1)$ interactions per unit of time, the (discrete) *parallel time* in a population protocol is defined as the number of interactions to occur, divided by the population size n. A continuous time variant [25] gives each molecule a rate-1 Poisson clock, upon which it reacts with a randomly chosen other molecule. The time until the next interaction is an exponential random variable with expected value $1/n$, so these two models are equivalent up to a re-scaling of time, which by straightforward Chernoff bounds is negligible. The continuous-time variant turns out to coincide *precisely* with the Gillespie model restricted to population protocols, when the volume $v = n$.

The naïve simulation algorithm for population protocols implements the model straightforwardly, one (potentially null) reaction at a time, taking time $\Theta(\ell)$ to simulate ℓ reactions.

4.1.1. Related work on faster stochastic simulation of CRNs. One "speedup heuristic" changes the model entirely: the *deterministic* or *continuous mass-action* model of CRNs represents the amount of each species as a nonnegative real-valued *concentration* (average count per unit volume), defining polynomial ordinary differential equations (ODEs) with a unique trajectory [26]. Integrating the ODEs using standard numerical methods [48] is typically much faster than stochastic simulation. There is a technical sense in which this model is the large-count, large-volume limit of the discrete stochastic model [37], but the ODE model removes any stochastic effects and can have vastly different behavior than the stochastic model [24, Fig. 3].

The obvious way to pick the next reaction to occur in the Gillespie algorithm, if there are R types of reactions, uses time $O(R)$. There are variants of Gillespie that reduce the time to apply a single reaction from $O(R)$ to $O(\log R)$ [28] or even $O(1)$ [56]. However, the time to apply ℓ total reactions still scales linearly with ℓ. Other exact methods have been developed that, for some CRNs, empirically appear to simulate ℓ reactions in time $o(\ell)$ [13, 45], but none have been proven rigorously to give an asymptotic speedup on all CRNs while maintaining exactness. Linear noise approximation [15] is another workaround, adding stochastic noise to an ODE trajectory.

Other methods give a provable and practical speedup, but they also provably sacrifice exactness, i.e., the distribution of trajectories sampled is not the same as the Gillespie algorithm. A common speedup heuristic for simulating $\omega(1)$ reactions in $O(1)$ time is *τ-leaping* [14, 31, 32, 50, 57], which "leaps" ahead by time τ, by assuming reaction propensities will not change and updating counts in a single batch step by sampling according these propensities. Such methods necessarily approximate the kinetics inexactly, though it is sometimes possible to prove bounds on the approximation accuracy [57]. Nevertheless, there are CRNs with stochastic effects not observed in ODE simulation, and where τ-leaping introduces systematic inaccuracies that disrupt the fundamental qualitative behavior of the system, demonstrating the need for exact stochastic simulation even with large population sizes; see [24, 38] for examples.

A speedup idea for population protocol simulation is to sample the number of each reaction that would result from a random matching of size ℓ (i.e., ℓ pairs of molecules, all disjoint), and update species counts in a single step. This heuristic, too, is inexact: unlike the true process, it prevents any molecule from participating in more than one of the next ℓ reactions.

However, based on this last heuristic, Berenbrink, Hammer, Kaaser, Meyer, Penschuck, and Tran [7] created a remarkable "batching" algorithm that is quadratically faster than the naïve population protocol simulation algorithm, yet samples from the same exact distribution. It can simulate, on population size n, batches of $\Theta(\sqrt{n})$ reactions in time $O(\log n)$ each. Conditioned on the event that no molecule is picked twice during the next ℓ interactions, these interacting pairs are a random disjoint matching of the molecules. Define the random variable $\mathbf{L}$ as the number of interactions until the same molecule is picked twice. The algorithm samples this "collision" length $\mathbf{L}$, then updates counts assuming all pairs of interacting molecules are disjoint until this collision, and finally simulates the interaction involving the collision. By the Birthday Paradox, $\mathrm{E}[\mathbf{L}] \approx \sqrt{n}$ in a population of n molecules, giving a quadratic factor speedup over the naïve algorithm. (The time to update a batch scales quadratically with the total number of species.)

The batching algorithm extends straightforwardly to a generalized model of population protocols in which, for some $o \in \mathbb{N}^+$, all reactions have exactly o reactants and o products (we call these *uniformly conservative* CRNs; see Section 4.2.2). However, the algorithm's correctness relies crucially on the fact that the population size n never changes. Reactions such as $A + B \to C$ that decrease n are easy to handle by adding inert "waste" products ($A + B \to C + W$) to make the

reaction conservative. The key challenge is reactions such as $A \to B + C$ that increase n; this is our focus.

4.1.2. Our contribution. We extend the batching algorithm of Berenbrink, Hammer, Kaaser, Meyer, Penschuck, and Tran [7] to the fully general model of CRNs. This is, to our knowledge, the first simulation algorithm for the CRN model provably preserving the exact stochastic dynamics, yet simulating ℓ reactions in time provably sublinear in ℓ. We implement our algorithm as a Python package, with remarkably fast performance in practice. See Section 5.1 for simulation data generated using this tool.

Our algorithm works on arbitrary CRNs and gives similar asymptotic speedup with respect to population size to the original batching algorithm for population protocols [7]. Our main theorem, Theorem 4.5.1, describes this speedup in detail.

There is an important efficiency difference between the Gillespie algorithm and the batching algorithms (both ours and [7]): the Gillespie algorithm automatically skips null reactions. For example, a reaction such as $L + L \to L + F$ in volume n, when $\#L = 2$ and $\#F = n - 2$, is much more efficient in the Gillespie algorithm, which simply increments the time until the $L + L \to L + F$ reaction by a sampled exponential random variable, in a single step taking time $O(1)$. A naïve population protocol simulation, on the other hand, iterates through $\Theta(n)$ expected null interactions ($L + F \to L + F$ and $F + F \to F + F$) until the two L's react. Even the batching algorithm, since it explicitly simulates null reactions, takes time $\Theta(\sqrt{n})$, in this regime still asymptotically (and practically) much slower than the Gillespie algorithm. For this reason, any practical implementation should switch to the Gillespie algorithm when most sampled reactions in a batch are null. We describe how many null reactions our algorithm tends to simulate in this way in Definition 4.2.8.

4.1.3. High-level overview of algorithm. The key insight to our extension of the batching algorithm to handle growing population sizes is that we modify the simulated CRN so that every reaction in the modified CRN increases the population size by the same amount (the *generativity* of the reaction). Furthermore, every reaction has the same number of reactants (the *order* of the reaction) in the modified CRN. The full transformation is described in Section 4.3. A simple example is the reversible reaction $A \underset{1}{\overset{1}{\rightleftharpoons}} B + C$; applying the transformation of Section 4.3 in volume

v results in the CRN, with two new species K and W, with reactions $A + K \xrightarrow{v/\#K} B + C + K$[1] and $B + C \xrightarrow{1} A + 2W$. More generally, add K as catalysts to appropriate reactions, until every reaction has the same order, then add W as products until every reaction has the same generativity. Note W ("waste") is never a reactant so does not affect reaction propensities; after each batch, W can be removed to prevent the population size from exploding.

Since every reaction has the same order, we can sample reactions in a batch with the same relative probabilities as the original reactions they represent. When we modify a reaction to have a K catalyst, we adjust its rate constant to compensate; the reaction retains the same probability of being sampled as in the original CRN. Since each reaction has the same generativity, the distribution of collision-free run lengths does not depend on which reactions are part of the run, allowing our batching algorithm to sample from this exact distribution before sampling which reactions occur.

In Section 4.4, we describe sampling in *discrete time*, i.e., drawing from the distribution of configurations after exactly ℓ reactions. This includes our core batching algorithm. Surprisingly, it is significantly more difficult to sample exactly in *continuous time*, i.e., from the distribution of configurations after exactly t time has elapsed. We describe this in Section 4.5, modifying the discrete-time algorithm and using adaptive rejection sampling [29] to efficiently sample inter-reaction times.

4.2. Definitions

4.2.1. Preliminaries. We use Λ to denote a generic finite set of labels for chemical species. We use $\mathbb{N}^\Lambda$ to denote the set of functions $f : \Lambda \to \mathbb{N}$. We can also interpret such a function as a vector $\mathbf{r}$ indexed by elements of Λ, calling $\mathbf{r}(S)$ the *count* of $S \in \Lambda$ in $\mathbf{r}$. Equivalently, it can be interpreted as a multiset, containing $\mathbf{r}(S)$ copies of each $S \in \Lambda$. We use $\|\mathbf{r}\|_1 = \sum_{A\in\Lambda} \mathbf{r}(A)$ or simply $\|\mathbf{r}\|$ to denote its 1-norm. We use $\mathbb{N}^\Lambda_i$ to denote the set of $\mathbf{r}$ such that $\|\mathbf{r}\| = i$. For $A \in \Lambda$ and $n \in \mathbb{N}$, we use $n \cdot A$ to denote the vector $\mathbf{a}$ with $\mathbf{a}(A) = n$ and $\mathbf{a}(B) = 0$ for all $B \neq A$. For $a, b \in \mathbb{N}$ with $a \geq b - 1$, we write $a^{\underline{b}}$ to denote the *falling power* $a!/(a-b)! = a(a-1)(a-2)\dots(a-b+1)$.

Given a set Λ, a *reaction* over Λ is a triple $\alpha = (\mathbf{r}, \mathbf{p}, k) \in \mathbb{N}^\Lambda \times \mathbb{N}^\Lambda \times \mathbb{R}_{>0}$, with $\mathbf{r}$ and $\mathbf{p}$ specifying the stoichiometric coefficients of the reactants and products, and k specifying the *rate constant*. A *chemical reaction network (CRN)* is a pair $\mathcal{C} = (\Lambda, R)$, where Λ is a finite set of chemical species

[1]Note K is a *catalyst*, both a reactant and product, so $\#K$ never changes; we set $\#K = n$, the molecular count, updating $\#K$ (and adjusting rate constants) between batches if needed to maintain $\#K = \Theta(n)$.

and R is a set of reactions over Λ. A *configuration* of a CRN is a vector $\mathbf{c} \in \mathbb{N}^{\Lambda}$. For each species $A \in \Lambda$, $\mathbf{c}(A)$ gives the count of A in $\mathbf{c}$.

Given a configuration $\mathbf{c}$ of a CRN $\mathcal{C} = (\Lambda, R)$, we say a reaction $\alpha = (\mathbf{r}, \mathbf{p}, k)$ is *applicable* if $\mathbf{r} \leq \mathbf{c}$, that is if there is enough of each reactant of α to carry out the reaction. If no reaction is applicable to $\mathbf{c}$, then we say $\mathbf{c}$ is *terminal*. If α is applicable to $\mathbf{c}$, we write $\alpha(\mathbf{c})$ to denote the configuration $\mathbf{c} - \mathbf{r} + \mathbf{p}$ obtained by subtracting the reactants and adding the products of α to $\mathbf{c}$. If $\mathbf{d} = \alpha(\mathbf{c})$ for some reaction $\alpha \in R$, then we write $\mathbf{c} \xrightarrow{\alpha} \mathbf{d}$ or simply $\mathbf{c} \to \mathbf{d}$. An *execution* $\mathcal{E}$ of $\mathcal{C}$ is a finite or infinite sequence of configurations $(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \ldots)$ such that for each $i \geq 0$, $\mathbf{c}_i \to \mathbf{c_{i+1}}$.

Given a CRN $\mathcal{C}$ and a configuration $\mathbf{c}$ of $\mathcal{C}$, for this chapter, the evolution of $\mathbf{c}$ over time is governed by the laws of *Gillespie kinetics* [30]. Each reaction $(\mathbf{r}, \mathbf{p}, k)$ is treated as a Poisson process whose rate is given by its *propensity*, defined for each reaction $\alpha = (\mathbf{r}, \mathbf{p}, k)$ in a configuration $\mathbf{c}$, with volume v, as

$$p_{\mathbf{c},v}(\alpha) = \frac{k}{v^{\|\mathbf{r}\|-1}} \cdot \prod_{A \in \mathbf{r}} \frac{\mathbf{c}(A)^{\underline{\mathbf{r}(A)}}}{\mathbf{r}(A)!}. \tag{11}$$

For example, if α is $A + 3B \xrightarrow{4.5} C$, then $p_{\mathbf{c},v}(\alpha) = \frac{4.5}{6v^3}\mathbf{c}(A)\mathbf{c}(B)(\mathbf{c}(B)-1)(\mathbf{c}(B)-2)$. The final product term of (11) ($\mathbf{c}(A)\mathbf{c}(B)(\mathbf{c}(B)-1)(\mathbf{c}(B)-2)$ in this example) is the number of ways to choose individual molecules from $\mathbf{c}$ with the given reactant identities. Note that reactions lacking sufficient reactants (e.g., if $\mathbf{c}(A) = 0$ or $\mathbf{c}(B) < 3$) have propensity 0. We also note that propensity is sometimes defined without this $\mathbf{r}(A)!$ factor; these two conventions are equivalent, and conversion between them is done by multiplying rate constants by a simple correction factor. Define the *total propensity* to be the sum of all propensities $p^{\text{tot}}_{\mathbf{c},v} = \sum_{\alpha \in R} p_{\mathbf{c},v}(\alpha)$. The next reaction to be executed is α with probability $\frac{p_{\mathbf{c},v}(\alpha)}{p^{\text{tot}}_{\mathbf{c},v}}$, and the elapsed time until this reaction occurs is distributed as an exponential random variable with rate $p^{\text{tot}}_{\mathbf{c},v}$.

The evolution of a CRN under Gillespie kinetics in some volume v can be viewed as a *continuous-time Markov chain* $\mathcal{M}_{\mathcal{C},v} = (S, \mathbf{R})$. The set of states S is the set of configurations of $\mathcal{C}$. The rate matrix $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is defined by taking, for every $(\mathbf{c}, \mathbf{d})$ such that $\mathbf{c} \to \mathbf{d}$,

$$\mathbf{R}(\mathbf{c}, \mathbf{d}) = \sum_{\{\alpha \in R | \mathbf{c} \xrightarrow{\alpha} \mathbf{d}\}} p_{\mathbf{c},v}(\alpha),$$

with $\mathbf{R}(\mathbf{c}, \mathbf{d}) = 0$ if no such reaction exists. The *transition matrix* $\overline{\mathbf{R}}$ is obtained by normalizing each row of $\mathbf{R}$ to have sum 1, so that (viewed as a matrix) the row $\overline{\mathbf{R}}_{\mathbf{c},\cdot}$ gives the probability of transition from $\mathbf{c}$ to each other configuration. If $\mathbf{c}$ is terminal, then $\overline{\mathbf{R}}(\mathbf{c}, \mathbf{c}) = 1$ and all other values in the row are 0.

4.2.2. Uniform CRNs.

Definition 4.2.1. *The* order $\mathrm{ord}(\alpha)$ *of a reaction* $\alpha = (\mathbf{r}, \mathbf{p}, k)$ *is* $\|\mathbf{r}\|$. *The* order $\mathrm{ord}(\mathcal{C})$ *of a CRN* $\mathcal{C}$ *is the maximum order of its reactions. If every reaction has the same order,* $\mathcal{C}$ *has* uniform order.

Definition 4.2.2. *The* generativity $\mathrm{gen}(\alpha)$ *of a reaction* $\alpha = (\mathbf{r}, \mathbf{p}, k)$ *is* $\|\mathbf{p}\| - \|\mathbf{r}\|$. *The* generativity $\mathrm{gen}(\mathcal{C})$ *of a CRN* $\mathcal{C}$ *is the maximum generativity of its reactions. If every reaction has the same generativity,* $\mathcal{C}$ *has* uniform generativity.

Definition 4.2.3. *A CRN is* uniform *if it has uniform order and uniform generativity; that is, there are* $r, p \in \mathbb{N}$ *such that every reaction has exactly* r *reactants and* p *products.*

Uniform CRNs are general enough to allow any CRN to be transformed into a uniform CRN with the same dynamics, while being constrained enough to allow batching. Previous work [43] defines and discusses *conservative* CRNs, which require $\|\mathbf{r}\| = \|\mathbf{p}\|$ for each *individual* reaction $(\mathbf{r}, \mathbf{p}, k)$, but allow different values of $\|\mathbf{r}\|$ between different reactions. We call a CRN that is both uniform and conservative *uniformly conservative*. (Population protocols are uniformly conservative with order 2.) Because uniform CRNs have uniform generativity, their molecular count changes in a predictable way, allowing similar analytical utility to conservative CRNs. Uniform order is also important for the batching algorithm, which is designed to simulate uniform CRNs. Thus the first step of our simulation algorithm is to translate any CRN into a uniform CRN; the next few definitions describe some quantities that need to be tracked while we do this transformation in order to preserve the dynamics of the original CRN. We also note that although $\mathrm{gen}(\mathcal{C})$ may be negative, this case is straightforward to handle with the original batching algorithm [7] by adding inert waste products to each reaction α with $\mathrm{gen}(\alpha) < \mathrm{gen}(\mathcal{C})$ until the CRN is uniformly conservative. Thus throughout the rest of our arguments we will assume $\mathrm{gen}(\mathcal{C}) \geq 0$ for any CRN $\mathcal{C}$.

4.2.3. Definitions for CRN transformation.

Definition 4.2.4. *Given a CRN $\mathcal{C} = (\Lambda, R)$, its* total rate constant *for a multiset of reactants $\mathbf{r} \in \mathbb{N}^\Lambda$, denoted $k^{\text{tot}}_{\mathcal{C}}(\mathbf{r})$, is given by*

$$k^{\text{tot}}_{\mathcal{C}}(\mathbf{r}) = \sum_{(\mathbf{r},\mathbf{p},k)\in R} k,$$

i.e., $k^{\text{tot}}_{\mathcal{C}}(\mathbf{r})$ is the sum of the rate constants of all reactions in R sharing the same reactants $\mathbf{r}$.

Definition 4.2.5. *A uniform CRN $\mathcal{C}$ is* uniformly reactive *if $k^{\text{tot}}_{\mathcal{C}}(\mathbf{r}) = k^{\text{tot}}_{\mathcal{C}}(\mathbf{r}')$ for all $\mathbf{r}, \mathbf{r}' \in \mathbb{N}^\Lambda_{\text{ord}(\mathcal{C})}$.*

In other words, for a uniformly reactive CRN, the sum of rate constants among all reactions sharing a reactant vector is constant, no matter which reactant vector is chosen. For example, the reactions

$$A \xrightarrow{1} B + C \qquad B \xrightarrow{2.5} A + C \qquad C \xrightarrow{1.5} A + A$$
$$A \xrightarrow{2} A + B \qquad B \xrightarrow{0.5} C + C$$

are not uniformly reactive because the sum of rate constants for reactions with A as the sole reactant is 3, same as for reactions with sole reactant B, yet it is 1.5 for C. But if we add the reaction $C \xrightarrow{1.5} B + B$, the CRN becomes uniformly reactive.

Uniformly reactive CRNs with order o have the useful property that the probability of choosing a particular multiset of reactants $\mathbf{r}$ for the next reaction under Gillespie kinetics (recall Eq. (11)) is *exactly* the probability that if we pick o molecules from the population uniformly at random without replacement, their identities are given by $\mathbf{r}$. In other words, picking reactants "Gillespie-style" is equivalent to picking reactants "population-protocol-style".

Lemma 4.2.6. *Let $\mathcal{C} = (\Lambda, R)$ be a uniformly reactive CRN and $\mathbf{c}$ be some configuration of $\mathcal{C}$. Let $\mathbf{X}$ denote the distribution of reactant vectors in $\mathbb{N}^\Lambda_{\text{ord}(\mathcal{C})}$ sampled by the Gillespie algorithm on $\mathbf{c}$ in volume v. Let $\mathbf{Y}$ denote the distribution of multisets of k molecules from $\mathbf{c}$, drawn by uniformly picking individual molecules without replacement. Then $\mathbf{X} = \mathbf{Y}$.*

Proof. The probability of the next reaction being some $\alpha \in R$ is the propensity of α divided by the total propensity of all reactions. Thus, in $\mathbf{X}$, the probability of the next reactant vector being $\mathbf{r}$ is proportional to the sum of the propensities of all α with that reactant vector, that is, to

$k_{\mathcal{C}}^{\text{tot}}(\mathbf{r})$. That is to say,

$$\begin{aligned}\Pr(\rho(\mathbf{c}) = \mathbf{r}) &= \frac{1}{\sum_{\beta \in R} p_{\mathbf{c},v}(\beta)} \sum_{\alpha=(\mathbf{r},\_,\_)\in R} p_{\mathbf{c},v}(\alpha) \\ &= \frac{k_{\mathcal{C}}^{\text{tot}}(\mathbf{r})}{\sum_{\beta \in R} p_{\mathbf{c},v}(\beta)} \cdot \prod_{A \in \mathbf{r}} \frac{\mathbf{c}(A)^{\underline{\mathbf{r}(A)}}}{\mathbf{r}(A)!}.\end{aligned}$$

By uniform reactivity, the fraction on the left is constant across choices of $\mathbf{r}$. The remainder then combinatorially gives exactly the number of ways to choose an ordered list of molecules from $\mathbf{c}$ that comprise $\mathbf{r}$ in some order. In $\mathbf{Y}$, the probability of $\mathbf{r}$ being drawn uniformly at random is also proportional to this quantity, so the two distributions must be identical. □

In transforming a CRN into a uniform one and modifying the order of reactions, any reaction with increased order will have its propensity change as a result of new reactants and the impact of volume on propensity. We give a way to relate this new propensity to the original propensity.

Definition 4.2.7. *Let $\mathcal{C} = (\Lambda, R)$ be a CRN, n be the molecular count of some configuration, and $v \in \mathbb{R}_{>0}$ be volume. Then for $\alpha = (\mathbf{r}, \mathbf{p}, k) \in R$, the* order-adjusted rate constant *for α is given by*

$$\bar{k}_{\mathcal{C}}(\alpha, n, v) = k \cdot \frac{v^{\delta_o}}{n^{\underline{\delta_o}}},$$

Where $\delta_o = \text{ord}(\mathcal{C}) - \text{ord}(\mathbf{r})$. Given a configuration $\mathbf{c}$ with $\|\mathbf{c}\| = n$, the order-adjusted propensity *of α is given by using its order-adjusted rate constant instead of k in Eq. (11), that is, by*

$$\bar{p}_{\mathbf{c},v}(\alpha) = \frac{\bar{k}_{\mathcal{C}}(\alpha, n, v)}{k} p_{\mathbf{c},v}(\alpha) \left(= \frac{k \cdot v^{\text{ord}(\mathcal{C})-1}}{n^{\underline{\delta_o}}} \cdot \prod_{A \in \mathbf{r}} \mathbf{c}(A)^{\underline{\mathbf{r}(A)}} \right).$$

This also allows us to precisely describe one potential source of inefficiency. Because we choose reactants instead of choosing reactions directly, we will sometimes choose reactants that have no corresponding reaction. We may also choose reactants which have low probability of reacting in the original CRN due to small rate constants, requiring an artificial slowdown to maintain exactness. We account for both of these effects together.

Definition 4.2.8. *Let $\mathcal{C} = (\Lambda, R)$ be a CRN, $\mathbf{c}$ be a configuration of $\mathcal{C}$ with $\|\mathbf{c}\| = n$, and $v \in \mathbb{R}_{>0}$ be volume. Let $\bar{p}^{\mathrm{tot}}_{\mathbf{c},v}$ be the total adjusted propensity of $\mathcal{C}$,*

$$\bar{p}^{\mathrm{tot}}_{\mathbf{c},v} = \sum_{\alpha \in R} \bar{p}_{\mathbf{c},v}(\alpha).$$

Let $\bar{k}^{\max}$ be the maximal adjusted rate constant of $\mathcal{C}$,

$$\bar{k}^{\max} = \max_{\alpha \in R} \bar{k}_{\mathcal{C}}(\alpha, n, v).$$

The scheduler slowdown factor *of $\mathcal{C}$ on $\mathbf{c}$ in volume v, denoted $S_{\mathcal{C},v}(\mathbf{c}) \geq 1$, is given by*

$$S_{\mathcal{C},v}(\mathbf{c}) = \frac{\bar{k}^{\max} \cdot \binom{2n}{\mathrm{ord}(\mathcal{C})}}{\bar{p}^{\mathrm{tot}}_{\mathbf{c},v}}.$$

Note that the term $2n$ is because n is the count of molecules before we add K; since we add n copies of K, the total count after this step is $2n$. Our algorithm can efficiently simulate any CRN execution whose configurations maintain a constant upper bound on this value. It is sufficient for this that *some* reaction with a large (i.e., not asymptotically smaller than any other) adjusted rate constant has no bottlenecks (i.e., all its reactants have count $\Omega(n)$). We note that it is possible for this condition to be true at some point in the simulation and become false later; for instance the reaction $2L \to L + F$ starts (with only L present) with $S_{\mathcal{C},v}(\mathbf{c}) \approx 4$ (due to unnecessary added K), but it increases as L is converted to F, eventually reaching $\Theta(n^2)$ when $\#F \gg \#L$, because most interactions are between two F's and are therefore passive.

4.2.4. Definitions relating to CRN simulation.

Definition 4.2.9. *Let $v \in \mathbb{R}_{>0}$ be volume. Let $\mathcal{C} = (\Lambda, R)$ be a CRN with corresponding continuous time Markov chain $\mathcal{M}_{\mathcal{C},v} = (\mathbb{N}^{\Lambda}, \mathbf{R})$. Let $\mathbf{c}$ be a configuration of $\mathcal{C}$, and $\ell \in \mathbb{N}$. Then the* discrete ℓ-future distribution *of $\mathcal{C}$, denoted $\mathcal{F}^{\mathrm{dis}}_{\mathcal{C},v}(\mathbf{c}, \ell)$, is given by $\overline{\mathbf{R}}^{\ell}_{\mathbf{c},\cdot}$. That is, it is the distribution of configurations of $\mathcal{C}$ starting from $\mathbf{c}$ after ℓ transitions (allowing terminal configurations to self-transition).*

The continuous t-future distribution *of $\mathcal{C}$, denoted $\mathcal{F}^{\mathrm{con}}_{\mathcal{C},v}(\mathbf{c}, t)$, is a distribution over $\mathbb{N}^{\Lambda} \cup \{\emptyset\}$ such that if $\mathbf{X} \sim \mathcal{F}^{\mathrm{con}}_{\mathcal{C},v}(\mathbf{c}, t)$, then for any configuration $\mathbf{d}$, $\Pr(\mathbf{X} = \mathbf{d})$ gives the probability of being in configuration $\mathbf{d}$ after t time starting from configuration $\mathbf{c}$, and $\Pr(\mathbf{X} = \emptyset)$ gives the probability*

that $\mathcal{C}$ is not in any valid configuration at time t (e.g., for CRNs, that infinitely many reactions happen by time t).

Definition 4.2.10. *A CRN $\mathcal{C}$ is said to exhibit* finite-time blowup *starting from some configuration* $\mathbf{c}$ *if* $\Pr(\mathcal{F}^{\text{con}}_{\mathcal{C},v}(\mathbf{c},t)=\emptyset)>0$ *for some* $v>0$ *and* $t>0$.

Finite-time blowup has been discussed for chemical systems governed by ODEs [36, 65]. It can also occur in stochastic CRNs. For example, the CRN $2A \xrightarrow{1} 3A$ exhibits finite-time blowup from any configuration with at least $2A$, in the sense that, for any time $t>0$, there is a positive probability that an infinite number of reactions occur before time t.

We now give a definition of simulation between CRNs. Many definitions of CRN simulation have been given in the literature. Different definitions are relevant in different contexts, such as in the continuous ODE model [16] or for formal verification of correspondence between different CRN implementations [35, 55]. Our definition of simulation is much simpler, because we do not need to worry about checking whether an arbitrary CRN simulates another arbitrary CRN. We are only worried about showing that any CRN can be simulated by its transformed version as shown in Section 4.3; the transformed CRN is closely related to the original. To this end, our only concern is that the CRNs have matching dynamics: that is, that the output of the Gillespie algorithm would look the same between them, ignoring the extra species introduced in the transformed CRN.

Definition 4.2.11. *Let $\mathcal{C}=(\Lambda',R)$ be a CRN and let $\Lambda\subseteq\Lambda'$. Given a configuration $\mathbf{c}$ of $\mathcal{C}$, define its* restriction to Λ, *denoted $\mathbf{c}|_\Lambda$, as the configuration obtained by taking $\mathbf{c}$ and setting the count of all species not in Λ to 0.*

Definition 4.2.12. *Let $v\in\mathbb{R}_{>0}$ be volume. Let $\mathcal{C}=(\Lambda,R)$ and $\mathcal{C}'=(\Lambda',R')$ with $\Lambda\subseteq\Lambda'$. We say* $\mathcal{C}'$ simulates $\mathcal{C}$ from $\mathbf{c}$ *if there exists a configuration $\mathbf{c}'$ of $\mathcal{C}'$ such that for all $t\geq 0$,*

$$\mathcal{F}^{\text{con}}_{\mathcal{C}',v}(\mathbf{c}',t)|_\Lambda = \mathcal{F}^{\text{con}}_{\mathcal{C},v}(\mathbf{c},t).$$

We say that $\mathcal{C}'$ conditionally simulates $\mathcal{C}$ from $\mathbf{c}$ *if this statement holds conditioned on these configurations being defined, that is, if $\mathbf{X}=\mathcal{F}^{\text{con}}_{\mathcal{C}',v}(\mathbf{c}',t)|_\Lambda$ and $\mathbf{Y}=\mathcal{F}^{\text{con}}_{\mathcal{C},v}(\mathbf{c},t)$, simulation requires that $\mathbf{X}=\mathbf{Y}$ while conditional simulation requires that*

$$(\mathbf{X}\mid\mathbf{X}\neq\emptyset)=(\mathbf{Y}\mid\mathbf{Y}\neq\emptyset).$$

If $\mathcal{C}'$ (conditionally) simulates $\mathcal{C}$ from every configuration $\mathbf{c}$ of $\mathcal{C}$, we say that $\mathcal{C}'$ (conditionally) simulates $\mathcal{C}$. We may also say $\mathcal{C}'$ simulates $\mathcal{C}$ in continuous time*; if any of the above statements hold for $\mathcal{F}^{\text{dis}}_{\mathcal{C},v}(\mathbf{c},\ell)$ and $\mathcal{F}^{\text{dis}}_{\mathcal{C}',v}(\mathbf{c}',\ell)$ (rather than $\mathcal{F}^{\text{con}}_{\mathcal{C},v}(\mathbf{c},t)$ and $\mathcal{F}^{\text{con}}_{\mathcal{C}',v}(\mathbf{c}',t)$), then we say $\mathcal{C}'$* simulates $\mathcal{C}$ in discrete time.

4.3. CRN transformations

In [7], working in the model of population protocols, the authors show how to simulate $\Theta(\sqrt{n})$ interactions on a population of n agents in $\Theta(\log n)$ time. The key insight is that sequential interactions between non-overlapping pairs of agents can be simulated simultaneously, as a *batch*. The results of these interactions only need to be computed once a *collision* occurs, i.e., an agent is chosen to interact a second time. On average $\Theta(\sqrt{n})$ such interactions will occur before a collision occurs. These interactions are referred to as a *collision-free run*. In particular, rather than simulating reactions individually, the authors first draw from the distribution of the length of a collision-free run. Then, they determine how many times each particular kind of interaction occurs within that run. Finally, they simultaneously apply the results of all of these interactions, along with one extra collision interaction sampled to include at least one agent that was part of the collision-free run.

Because general CRNs may contain reactions which unpredictably change total molecular count, this idea is not immediately applicable. However, this methodology can be applied to uniform CRNs, with some modifications to account for changing molecular count and reactions of arbitrary uniform order (as opposed to uniform order 2) Thus our first step is to transform an arbitrary CRN $\mathcal{C}$ into a uniform CRN simulating $\mathcal{C}$ (see Section 4.2.4 for our definition of CRN simulation).

4.3.1. Transforming an arbitrary CRN into a uniform CRN. Given an arbitrary CRN $\mathcal{C}$, we transform it into a uniform CRN $\mathcal{C}'$ via Algorithm 1. We will demonstrate how this method works on the example CRN,

$$A + B \xrightarrow{2} C$$

$$C \xrightarrow{3} A + B$$

with a single reversible dimerization reaction, a case that existing methods cannot simulate efficiently. First, the second reaction's order is one less than the order of the CRN, so we add one

Algorithm 1 Uniform CRN transformation

Input: CRN $\mathcal{C} = (\Lambda, R)$, volume $v \in \mathbb{R}_{>0}$, integer $k_0 \geq \text{ord}(\mathcal{C})$
Output: Uniform CRN $\mathcal{C}' = (\Lambda', R')$ simulating $\mathcal{C}$ as in Lemma 4.3.1

(1) Add two new species, K and W, setting $\Lambda' = \Lambda + \{K, W\}$.
(2) To define R', for each reaction $\alpha = (\mathbf{r}, \mathbf{p}, k)$ in R, let $\delta_o = \text{ord}(\mathcal{C}) - \text{ord}(\alpha)$ and $\delta_g = \text{gen}(\mathcal{C}) - \text{gen}(\alpha)$, and add the reaction $\alpha' = (\mathbf{r}', \mathbf{p}', k')$ to R', where
    (a) $\mathbf{r}' = \mathbf{r} + \delta_o \cdot K$,
    (b) $\mathbf{p}' = \mathbf{p} + \delta_o \cdot K + \delta_g \cdot W$,
    (c) $k' = \bar{k}_{\mathcal{C}}(\alpha, k_0, v)$ (see Definition 4.2.7).

    In other words: add $\text{ord}(\mathcal{C}) - \text{ord}(\alpha)$ copies of K to both $\mathbf{r}$ and $\mathbf{p}$, add $\text{gen}(\mathcal{C}) - \text{gen}(\alpha)$ copies of W to $\mathbf{p}$, and adjust k by a multiplicative factor depending on input constants and how many K were added to $\mathbf{r}$.

copy of K to catalyze the second reaction:

$$A + B \xrightarrow{2} C$$

$$C + K \xrightarrow{3} A + B + K$$

Next, the first reaction's generativity is -1, while the CRN has generativity 1. So we add 2 copies of W to the first reactant's products:

$$A + B \xrightarrow{2} C + W + W$$

$$C + K \xrightarrow{3} A + B + K$$

Finally, we must adjust the rate constant, accounting for volume and our choice of k_0. Here, this will result in multiplying the second reaction's rate constant by $\frac{v}{k_0}$, so if we take, for example, $v = 6$ and $k_0 = 30$, this would result in the output CRN:

$$A + B \xrightarrow{2} C + W + W$$

$$C + K \xrightarrow{0.6} A + B + K$$

Intuitively, W is an inert waste species and does not affect the dynamics of the CRN. K has constant count, so its presence has a constant effect on each reaction's propensity over the course of a batch, which is accounted for by modifying rate constants. k_0 represents the count of K that will be added to the original CRN configuration for simulation. We require $k_0 \geq \text{ord}(\mathcal{C})$ for simplicity. Choosing $k_0 \in \Theta(n)$ yields the simplest asymptotic analysis, and is a reasonable practical choice.

Lemma 4.3.1. *Let $\mathcal{C} = (\Lambda, R)$ be a CRN, and let $\mathcal{C}' = (\Lambda', R')$ be the output of Algorithm 1 on $\mathcal{C}$ with any choice of v and k_0. Then $\mathcal{C}'$ simulates $\mathcal{C}$ in continuous and discrete time.*

PROOF. Let $\mathbf{k} = k_0 \cdot K$. Given a configuration $\mathbf{c}$ of $\mathcal{C}$, we will use the configuration $\mathbf{c}' = \mathbf{c} + \mathbf{k}$ of $\mathcal{C}'$ to satisfy the definition of simulation given in Definition 4.2.12.

Consider the Markov chain $\mathcal{M}_{\mathcal{C}',v} = (S', \mathbf{R}')$ representing $\mathcal{C}'$. Define an equivalence relation $\sim$ on S' by $\mathbf{c}_1 \sim \mathbf{c}_2 \iff \forall A \in \Lambda' \setminus \{W\}, \mathbf{c}_1(A) = \mathbf{c}_2(A)$, i.e., they differ only in their count of W. Let $\widetilde{\mathcal{M}}_{\mathcal{C}'}$ be the continuous time Markov chain whose states are the equivalence classes under $\sim$ of the states of S' which contain exactly k_0 copies of K, with transitions inherited from $\mathbf{R}'$. This inheritance is well-defined because W is inert, so equivalent states always have transitions to equivalent states. Note also that these are the only equivalence classes we need to consider, as any configuration reachable from $\mathbf{c}'$ has k_0 copies of K.

We observe that $\widetilde{\mathcal{M}}_{\mathcal{C}'}$ is isomorphic to $\mathcal{M}_{\mathcal{C}}$. For any state $\mathbf{d}$ in $\mathcal{M}_{\mathcal{C}}$, the corresponding state in $\widetilde{\mathcal{M}}_{\mathcal{C}'}$ is the equivalence class of $\mathbf{d} + \mathbf{k}$. By construction, corresponding reactions have equal propensities in these two CRNs: the factor by which Algorithm 1 multiplies k to obtain k' accounts for the extra K in the reactants and the need to divide by a different power of v. Explicitly, for any pair of corresponding reactions $\alpha = (\mathbf{r}, \mathbf{p}, k)$ and $\alpha' = (\mathbf{r}', \mathbf{p}', k')$,

$$\begin{aligned}
p_{\mathbf{c}',v}(\alpha') &= \frac{k'}{v^{\|\mathbf{r}'\|}} \cdot \left(\prod_{A \in \mathbf{r}'} \mathbf{c}'(A)^{\underline{\mathbf{r}'(A)}} \right) \\
&= \frac{k \cdot \frac{v^{\mathbf{r}'(K)}}{\lfloor v \rfloor^{\underline{\mathbf{r}(K)}}}}{v^{\mathbf{r}'(K)+\|\mathbf{r}\|}} \cdot \left(\lfloor v \rfloor^{\underline{\mathbf{r}(K)}} \prod_{A \in \mathbf{r}} \mathbf{c}(A)^{\underline{\mathbf{r}(A)}} \right) \\
&= \frac{k}{v^{\|\mathbf{r}\|}} \cdot \left(\prod_{A \in \mathbf{r}} \mathbf{c}(A)^{\underline{\mathbf{r}(A)}} \right) \\
&= p_{\mathbf{c},v}(\alpha).
\end{aligned}$$

This shows that corresponding transitions have the same rates in $\mathcal{M}_{\mathcal{C}}$ and $\widetilde{\mathcal{M}}_{\mathcal{C}'}$. It follows that they have identical distributions of configurations sampled at any time or number of steps. □

4.3.2. Transforming a uniform CRN into a uniformly reactive CRN. Our batching algorithm is based on population protocols: it chooses a random set of molecules, rather than a random reaction. These molecules may not correspond to a reaction. In this case, for consistency,

we still need molecular count to update. Additionally, we must account for differing rate constants between different reactions.

We solve these issues by ensuring that the value $k^{\text{tot}}_{\mathcal{C}}(\mathbf{r})$ (see Definition 4.2.4) is identical among all every possible multiset $\mathbf{r}$ of reactants the batching algorithm might sample. This is the notion of a *uniformly reactive* CRN (Definition 4.2.5). Algorithm 2 transforms the output of Algorithm 1 into such a CRN. It does so by adding *passive reactions*, which update molecular count consistently but only add W, and thus do not affect any reaction's propensity.

For example, the example given for Algorithm 1 would be transformed into the following:

$$A + B \xrightarrow{2} C + W + W$$

$$C + K \xrightarrow{0.6} A + B + K$$

$$C + K \xrightarrow{1.4} C + K + W$$

$$A + A \xrightarrow{2} A + A + W$$

$$A + C \xrightarrow{2} A + C + W$$

$$\vdots$$

$$W + W \xrightarrow{2} W + W + W,$$

where there is a passive reaction with rate constant 2 for every possible reactant multiset except for $A + B$ and $C + K$.

Algorithm 2 Total rate constant uniformity transformation

Input: CRN $\mathcal{C} = (\Lambda, R)$, volume $v \in \mathbb{R}_{>0}$, integer $k_0 \geq \text{ord}(\mathcal{C})$
Output: Uniformly reactive CRN $\mathcal{C}'$ simulating $\mathcal{C}$ as in Lemma 4.3.3

(1) Let $\mathcal{C}_0 = (\Lambda + \{K, W\}, R_0)$ be the uniform CRN output by Algorithm 1 on $\mathcal{C}$, v and k_0.
(2) Let $k^{\max} = \max_{\mathbf{r} \in \mathbb{N}^{\Lambda}_{\text{ord}(\mathcal{C}_0)}} k^{\text{tot}}_{\mathcal{C}_0}(\mathbf{r})$, the maximum total rate constant of any set of reactants.
(3) Define a set of *passive reactions* P: for every $\mathbf{r} \in \mathbb{N}^{\Lambda}_{\text{ord}(\mathcal{C}_0)}$ such that $k^{\text{tot}}_{\mathcal{C}_0}(\mathbf{r}) < k^{\max}$, P contains the reaction
$$(\mathbf{r}, \mathbf{r} + \text{gen}(\mathcal{C}_0) \cdot W, k^{\max} - k^{\text{tot}}_{\mathcal{C}_0}(\mathbf{r})).$$
(4) Output $\mathcal{C}' = (\Lambda + \{K, W\}, R_0 + P)$.

Observation 4.3.2. *The output of Algorithm 2 is uniformly reactive (see Definition 4.2.5).*

Lemma 4.3.3. *Let $\mathcal{C} = (\Lambda, R)$ be a CRN, and $\mathbf{c}$ a configuration of $\mathcal{C}$. Let $\mathcal{C}' = (\Lambda', R')$ be the result of running Algorithm 2 on $\mathcal{C}$. Then $\mathcal{C}'$ conditionally simulates $\mathcal{C}$ in continuous and discrete time.*

Note that we can only guarantee conditional simulation because the reactions we add to $\mathcal{C}'$ may cause it to exhibit finite-time blowup. For example, the reaction $2W \to 3W$ causes finite-time blowup on its own. Removing W between batches prevents this issue.

Proof. Given a configuration $\mathbf{c}$ of $\mathcal{C}$, we simulate it from $\mathbf{c} + k_0 \cdot K$. We must show that these two CRNs have identical distributions over configurations at time t, conditioned on both of them having valid configurations at time t. By construction, there is a natural injection $f : R \to R'$ sending $\alpha \in R$ to its modified version in R' that was output by Algorithm 1. We couple the two CRNs as follows: whenever a reaction α' occurs in $\mathcal{C}'$, if there is some $\alpha \in R$ such that $f(\alpha) = \alpha'$, then α occurs in $\mathcal{C}$. Otherwise (i.e., if α' is passive), nothing happens in $\mathcal{C}$. This is a valid coupling because any passive α' only affect the count of W, so their occurrence does not affect the propensity of any reactions in R' nor the count of any species in Λ. Therefore, from the point of view of $\mathcal{C}$, these extra reactions do not affect the dynamics of the system so long as $\mathcal{C}'$ eventually reaches time t, which is guaranteed by conditioning. □

4.4. Simulating discrete-time CRN executions

We have shown that any CRN can be simulated by a uniformly reactive CRN. In this section, we show how to leverage this to efficiently sample a "discrete-time" CRN execution, meaning that we will merely count the number of reactions in each batch, but make no attempt to sample the amount of continuous time elapsed during the batch according to the Gillespie distribution. That is, given a CRN $\mathcal{C}$, a starting configuration $\mathbf{c}$, volume v, and a number of steps to simulate ℓ, we wish to efficiently draw from the distribution $\mathcal{F}^{\text{dis}}_{\mathcal{C},v}(\mathbf{c}, \ell)$ (Definition 4.2.12). Section 4.5 describes how to modify the algorithm to sample accurate timestamps to assign to each sampled configuration.

4.4.1. Discrete sampling with a scheduler. We now turn our attention to algorithms which simulate CRNs, rather than CRNs which simulate each other. The first such algorithm, Algorithm 3, is analogous to the algorithm SEQ in [7]. It is not intended to be executed, and is provided for analytical comparison. The only adjustment we must make is to convert rate constants into probabilities.

Algorithm 3 Scheduler-based uniform CRN simulation

Input: CRN $\mathcal{C} = (\Lambda, R)$, volume $v \in \mathbb{R}_{>0}$, configuration $\mathbf{c}$ of $\mathcal{C}$, integer $\ell \in \mathbb{N}$
Output: Configuration $\mathbf{c}'$ of $\mathcal{C}$ distributed as in Corollary 4.4.1

Let $\mathcal{C}'$ be the uniformly reactive CRN output by Algorithm 2 on input $\mathcal{C}$, v, and $k_0 = \|\mathbf{c}\|$. Let $\mathbf{c}' = \mathbf{c} + \|\mathbf{c}\| \cdot K$. Set `steps` $= 0$, and repeat until `steps` $= \ell$:

(1) Choose $\mathrm{ord}(\mathcal{C}')$ molecules from $\mathbf{c}'$, each uniformly at random without replacement, and let $\mathbf{r}$ be the resulting multiset.
(2) Choose a reaction α from $\mathcal{C}'$ with reactant vector $\mathbf{r}$, with probability proportional to its rate constant (i.e., reaction $\alpha_i = (\mathbf{r}_i, \mathbf{p}_i, k_i)$ is chosen with probability $\frac{k_i}{k^{\mathrm{tot}}_{\mathcal{C}'}(\mathbf{r})}$). Update $\mathbf{c}'$ by executing α.
(3) If a non-passive reaction was executed, (i.e., anything other than $\mathbf{r} \to \mathbf{r} + \mathrm{gen}(\mathcal{C}') \cdot W$), increment `steps`.
(4) Set $\mathbf{c}'(W)$ to 0 (to avoid finite-time blowup).

The next corollary follows directly from Lemmas 4.2.6 and 4.3.3.

Corollary 4.4.1. *On input $\mathcal{C}$, v $\mathbf{c}$, and ℓ, the output of Algorithm 3 is distributed as $\mathcal{F}^{\mathrm{dis}}_{\mathcal{C},v}(\mathbf{c}, \ell)$.*

As we will analyze our main algorithm using this algorithm as an intermediate, we require that it runs in time linear in ℓ. It may take longer if it repeatedly simulates passive reactions. We incorporate this into our analysis here.

Lemma 4.4.2. *Let $\mathcal{C}$ be a CRN, $\mathbf{c}$ be a configuration of $\mathcal{C}$, and $\ell \in \mathbb{N}$. Then the process of sampling from $\mathcal{F}^{\mathrm{dis}}_{\mathcal{C},v}(\mathbf{c}, \ell)$ using Algorithm 3 takes $O\left(S_{\mathcal{C},v}(\mathbf{c})\right)$ steps (see Definition 4.2.8). That is, each loop iteration executes a non-passive reaction with probability $\Omega(\frac{1}{S_{\mathcal{C},v}(\mathbf{c})})$.*

Proof. Let $\mathrm{n} = \|\mathbf{c}\|$. Consider how likely the Gillespie algorithm is to execute a passive reaction when running $\mathcal{C}'$ on $\mathbf{c}$. We claim this probability is given *exactly* by $\frac{1}{S_{\mathcal{C},v}(\mathbf{c})}$. The probability of one of a set of reactions being run under Gillespie kinetics is the sum of those reactions' propensities, divided by the total propensity. This is how $S_{\mathcal{C},v}(\mathbf{c})$ is defined: its denominator gives the total propensity of all reactions in $\mathcal{C}'$ that came from $\mathcal{C}$, while its numerator gives the total propensity of all reactions in $\mathcal{C}'$. This latter claim can be seen because we simulate $\mathcal{C}'$ on a configuration of size $2n$, and each set of $\mathrm{ord}(\mathcal{C})$ molecules in $\mathbf{c}$ will contribute a value of its corresponding total rate constant to the total propensity, and uniform reactivity implies that this value is the same for all $\binom{2n}{\mathrm{ord}(\mathcal{C})}$ of these sets. □

4.4.2. Batch size sampling. In [7], the authors use a notion of collision-free runs, and sample from the length of such a run. Our analysis will be similar to theirs, with some additional details to

account for arbitrary order and generativity. Below, we use "red" and "green" to refer to molecules that have interacted (respectively, not interacted) in a batch.

Definition 4.4.3. *Let $\mathcal{C}$ be a uniformly reactive CRN and $\mathbf{c}$ be a configuration of $\mathcal{C}$. Suppose each molecule in $\mathbf{c}$ is colored red or green, and that reactions consume specific copies of molecules uniformly at random. Whenever a reaction occurs, all products (including catalysts) are colored red. Then, given an execution $\mathcal{E}$ of $\mathcal{C}$ from $\mathbf{c}$, we say that the* collision index *of $\mathcal{E}$ is the index (starting from 0) of the first reaction with any red molecule as a reactant. That is, it is the number of reactions that are executed before any red molecule is a reactant.*

We define the random variable ℓ as the collision index of $\mathcal{E}$ when $\mathcal{E}$ is sampled (using normal stochastic sampling) among all (infinite) executions of $\mathcal{C}$ from $\mathbf{c}$. We say $\ell \sim \mathbf{coll}(n, r, o, g)$,[2] where $n = |\mathbf{c}|$, r is the count of red molecules in $\mathbf{c}$, $o = \mathrm{ord}(\mathcal{C})$ and $g = \mathrm{gen}(\mathcal{C})$.

Note that because $\mathcal{C}$ is uniformly reactive (recall Definition 4.2.5), this distribution will not depend on the specific reactions in $\mathcal{C}$ or species present in $\mathbf{c}$.

Definition 4.4.4. *The* multifactorial *$n!^{(g)}$ is given by the product $n(n-g)(n-2g)\ldots$, continuing until (and including) the last positive term.*

Lemma 4.4.5. *Let $\ell \sim \mathbf{coll}(n, r, o, g)$, where $g \geq 0$. Then ℓ has (reverse) cumulative distribution*

$$\Pr(\ell \geq k) = \begin{cases} \frac{(n-r)!}{(n-r-ko)!} \prod_{j=0}^{o-1} \frac{(n-g-j)!^{(g)}}{(n+g(k-1)-j)!^{(g)}} & 0 \leq k \leq \frac{n-r}{o}, g > 0 \\ \frac{(n-r)!}{(n-r-ko)!} \prod_{j=0}^{o-1} \frac{1}{(n-j)^k} & 0 \leq k \leq \frac{n-r}{o}, g = 0 \\ 0 & \textit{otherwise.} \end{cases}$$

Proof. Because of uniform reactivity, we may ignore what species each molecule is: at each step, every o-tuple of molecules is equally likely to comprise the next reaction's reactants. Therefore, we can equivalently consider a scheduler that repeatedly picks individual reactants without replacement, and executes a reaction every time it picks o of them (returning the products to the

[2]For our purposes, the parameter r will always be 0; however, we show the computation in generality, as general r is necessary to implement *multibatching* as described in [7]. Adapting our result to this regime can be done in much the same way as in [7].

pool of molecules it may pick). If, after a reaction, the population has m total molecules with l green and $m-l$ red, then the probability of this scheduler only picking green molecules in the *next* reaction is

$$\prod_{j=0}^{o-1} \frac{l-j}{m-j}.$$

Initially, there are n molecules, r red and $n-r$ green. After each reaction, o green molecules are replaced with $o+g$ red molecules. Therefore, the probability of executing at least k reactions before a collision is given by

$$\Pr[\ell \geq k] = \prod_{i=0}^{k-1} \prod_{j=0}^{o-1} \frac{\overbrace{n-r-oi}^{\#\text{green after } i \text{ rxns}} - j}{\underbrace{n+gi}_{\text{pop size after } i \text{ rxns}} - j}.$$

Examining the numerator and denominator separately, we observe that the numerator varies over all integers from $n-r-ko+1$ to $n-r$, i.e., the combined product of all numerators is the ratio $\frac{(n-r)!}{(n-r-ko)!}$. Factoring this out and commuting the products yields:

$$\Pr[\ell \geq k] = \frac{(n-r)!}{(n-r-ko)!} \cdot \prod_{j=0}^{o-1} \prod_{i=0}^{k-1} \frac{1}{n+gi-j}.$$

This inner product is the "Pochhammer g-symbol" $(n+gi-j)_{k,g}$.[3] Our task is now to rewrite it in a way that can be efficiently computed when k is large. When $g > 0$, we can rewrite the inner product as a ratio where all but k terms cancel by using multifactorials (Definition 4.4.4):

$$\frac{(n-r)!}{(n-r-ko)!} \cdot \prod_{j=0}^{o-1} \frac{(n-g-j)!^{(g)}}{(n+g(k-1)-j)!^{(g)}}.$$

When $g = 0$, corresponding to a uniformly conservative CRN, [4] each term in the inner product is identical and the formula simplifies to:

$$\frac{(n-r)!}{(n-r-ko)!} \cdot \prod_{j=0}^{o-1} \frac{1}{(n-j)^k}. \qquad \square$$

Lemma 4.4.6. *It is possible to sample from* $\mathbf{coll}(n, r, o, g)$ *in time* $O(o \cdot \log n)$.

[3] This is normally called the "Pochhammer k-symbol", but we use k as the first parameter rather than the second.
[4] Note that even in this case and when $o = 2$ (i.e., for population protocols), we obtain a different expression from [7], since we use a different definition of this distribution in terms of reactions (o molecules sampled at a time) instead of reactants (1 molecule sampled at a time).

PROOF. To sample $\ell \sim \mathbf{coll}(n, r, o, g)$, we can draw a uniform variate $U \sim \mathrm{Unif}([0, 1])$ and use inversion sampling [21] to draw a sample via binary search in $O(\log n)$ comparisons between U and the CDF (because $0 \leq \ell \leq n$). To compare U to the formula given by Lemma 4.4.5, we take the log of both sides and see that the relation we must compute is whether or not

$$\log((n-r)!) - \log((n-r-ko)!) + \sum_{j=0}^{o-1} \left[\log\left((n-g-j)!^{(g)}\right) - \log\left((n+g(k-1)-j)!^{(g)}\right)\right] < \log U.$$

To compute this efficiently, we make use of the Γ function, the generalization of factorial to non-integers. Computing $\log(x!)$ for any x can be done efficiently using standard implementations of the log-gamma function $\log(\Gamma(x))$. To compute the log of a multifactorial, we make use of the relation that $\Gamma(x+1) = x \cdot \Gamma(x)$, even for non-integer x. We divide each term of each multifactorial product by g, and collect these factors of g. This yields a product where consecutive terms differ by 1, which can be written as a ratio of Gamma functions. For example,

$$\begin{aligned}
\log(17!^{(5)}) &= \log(17 \cdot 12 \cdot 7 \cdot 2) = \log\left(5^4 \cdot \frac{17}{5} \cdot \frac{12}{5} \cdot \frac{7}{5} \cdot \frac{2}{5}\right) = \log\left(5^4 \cdot \frac{17}{5} \cdot \frac{\Gamma(17/5)}{\Gamma(17/5)} \cdot \frac{12}{5} \cdot \frac{7}{5} \cdot \frac{2}{5}\right) \\
&= \log\left(5^4 \cdot \frac{\Gamma(22/5)}{\Gamma(17/5)} \cdot \frac{12}{5} \cdot \frac{7}{5} \cdot \frac{2}{5}\right) = \log\left(5^4 \cdot \frac{\Gamma(22/5)}{\Gamma(17/5)} \cdot \frac{12}{5} \cdot \frac{\Gamma(12/5)}{\Gamma(12/5)} \cdot \frac{7}{5} \cdot \frac{2}{5}\right) \\
&= \log\left(5^4 \cdot \frac{\Gamma(22/5)}{\Gamma(17/5)} \cdot \frac{\Gamma(17/5)}{\Gamma(12/5)} \cdot \frac{7}{5} \cdot \frac{2}{5}\right) = \log\left(5^4 \cdot \frac{\Gamma(22/5)}{\Gamma(12/5)} \cdot \frac{7}{5} \cdot \frac{2}{5}\right) \\
\ldots &= \log\left(5^4 \cdot \frac{\Gamma(22/5)}{\Gamma(2/5)}\right) = 4\log(5) + \log\Gamma(22/5) - \log\Gamma(2/5).
\end{aligned}$$

Using this method, these log-multifactorial terms can be computed efficiently even if they contain many terms. It follows that every term on the left-hand side can be computed efficiently, and there are $\Theta(o)$ terms. □

Since we start each batch with no molecules having interacted, in the next lemma, we consider only this case ($r = 0$ in $\mathbf{coll}(n, 0, o, g)$); however, we strongly suspect that a similar result holds that was shown in [7], that for $r \geq \sqrt{n}$, $\mathrm{E}[\ell] = \Theta(n/r)$.

Lemma 4.4.7. *Let $\ell \sim \mathbf{coll}(n, 0, o, g)$. Then $\mathrm{E}[\ell] = \Theta(\sqrt{n})$.*

PROOF. Recall from the proof of Lemma 4.4.5 that (setting $r = 0$)

$$\Pr[\ell \geq k] = \prod_{i=0}^{k-1} \prod_{j=0}^{o-1} \frac{n - oi - j}{n + gi - j}.$$

We first show the lower bound:

$$\mathrm{E}[\ell] = \sum_{k=1}^{\lfloor n/o \rfloor} \Pr[\ell \geq k] = \sum_{k=1}^{\lfloor n/o \rfloor} \prod_{i=0}^{k-1} \prod_{j=0}^{o-1} \frac{n-oi-j}{n+gi-j} > \sum_{k=1}^{\lfloor n/o \rfloor} \prod_{i=0}^{k-1} \prod_{j=0}^{o-1} \frac{n-oi-(o-1)}{n+gi}$$

$$= \sum_{k=1}^{\lfloor n/o \rfloor} \prod_{i=0}^{k-1} \left(\frac{n-oi-o+1)}{n+gi}\right)^{o} > \sum_{k=1}^{\lfloor n/o \rfloor} \prod_{i=0}^{k-1} \left(\frac{n-o(k-1)-o+1}{n+g(k-1)}\right)^{o} = \sum_{k=1}^{\lfloor n/o \rfloor} \left(\frac{n-ok+1}{n+g(k-1)}\right)^{ok}$$

$$> \sum_{k=1}^{\lfloor n/o \rfloor} \left(\frac{n-ok}{n+gk}\right)^{ok} > \sum_{k=1}^{\sqrt{n}} \left(\frac{n-ok}{n+gk}\right)^{ok} = \sum_{k=1}^{\sqrt{n}} \left(\frac{n}{n+gk} - \frac{ok}{n+gk}\right)^{ok}$$

$$= \sum_{k=1}^{\sqrt{n}} \left(\frac{n+gk}{n+gk} - \frac{ok+gk}{n+gk}\right)^{ok} = \sum_{k=1}^{\sqrt{n}} \left(1 - \frac{ok+gk}{n+gk}\right)^{ok} > \sum_{k=1}^{\sqrt{n}} \left(1 - \frac{(o+g)\sqrt{n}}{n}\right)^{ok}$$

$$= \sum_{k=1}^{\sqrt{n}} \left(1 - \frac{o+g}{\sqrt{n}}\right)^{ok} > \sum_{k=1}^{\sqrt{n}} \left(1 - \frac{o(o+g)}{\sqrt{n}}\right)^{k} \qquad \text{Bernoulli's inequality}$$

$$(1-x)^o > 1 - xo \text{ when } x \leq 1$$

$$= \frac{1 - \left(1 - \frac{o(o+g)}{\sqrt{n}}\right)^{\sqrt{n}}}{1 - \left(1 - \frac{o(o+g)}{\sqrt{n}}\right)} = \sqrt{n}\frac{1 - \left(1 - \frac{o(o+g)}{\sqrt{n}}\right)^{\sqrt{n}}}{o(o+g)} \geq \sqrt{n}\frac{1-e^{-o(o+g)}}{o(o+g)}, \text{ since } \left(1 + \frac{x}{\sqrt{n}}\right)^{\sqrt{n}} \leq e^{x}.$$

For the upper bound, recall in the definition of $\ell \sim \mathbf{coll}(n, r, o, g)$ that g is the number of additional red molecules added each reaction, in addition to the o green molecules that are turned red. Clearly ℓ has a smaller expected value when $g = 0$ than when $g > 0$, since the extra red molecules in the latter case make it more likely that we pick a red reactant molecule on each interaction. Thus for upper-bounding $\mathrm{E}[\ell]$, we make the worst-case assumption that $g = 0$. Similar reasoning lets us assume $o = 1$ in the worst case for the upper bound. This turns out to be precisely what was analyzed in [7, Lemma 4],[5] where it is shown $\mathrm{E}[\ell] \leq 2\sqrt{n}$. □

4.4.3. Single batch content sampling. To determine what reactions to simulate in a batch, we also use a method very similar to that of [7]. In population protocols over q states, the authors view this problem as sampling the values of a $q \times q$ transition matrix D. To do this, they first sample the row sums of D (corresponding to the first reactant) using a multivariate hypergeometric distribution,[6] and then sample values within each row (corresponding, for each possible first

[5]This is in fact the distribution of the original birthday problem: how many balls can we throw into n bins before some bin gets two balls? Although [7] studies order-2 CRNs, their definition of a run length is based on the number of molecules that interact, as opposed to the number of interactions as in our definition.

[6]A multivariate hypergeometric distribution asks, if we sample ℓ molecules from an urn without replacement, prescribed initial counts of molecule species $\{1, \ldots, q\}$, how many of each species do we get?

reactant species r_1, how many of each species r_2 get paired as second reactant to r_1), taking $q + q^2$ samples. We use a similar approach, except that we must populate an $(\mathrm{ord}(\mathcal{C}))$-dimensional array D. We first sample the $q = |\Lambda|$ codimension-1 sums of D: that is, for each species, how many reactions in the batch will have that species as their first reactant. Once we have done this, we are left with q $(\mathrm{ord}(\mathcal{C}) - 1)$-dimensional subarrays, and can recursively use multivariate hypergeometric distributions until we have sampled all elements. This will require $\Theta(q^{\mathrm{ord}(\mathcal{C})})$ multivariate hypergeometric samples, each of which each can be obtained in constant time [60]. In distribution, this process is equivalent to sampling all the individual reactants for every reaction one at a time.

4.4.4. Full discrete-time simulation algorithm. With this, we are ready to describe the complete algorithm which efficiently samples from the discrete-time distribution $\mathcal{F}^{\mathrm{dis}}_{\mathcal{C},v}(\mathbf{c}, \ell)$. Algorithm 4 simulates one batch, while Algorithm 5 runs batches until it has simulated ℓ reactions.

Algorithm 4 Discrete-time single-batch simulation

Input: Uniformly reactive CRN $\mathcal{C}' = (\Lambda, R)$, configuration $\mathbf{c}_0$ of $\mathcal{C}'$, batch size bound $\ell_{\max} \in \mathbb{N}$
Output: Number of steps simulated $\ell_{\mathrm{out}} \leq \ell_{\max}$, Configuration $\mathbf{c}$ of $\mathcal{C}'$ distributed as in Lemma 4.4.8

(1) Let $\mathbf{c}'$ be an empty configuration, and let $\mathbf{c} = \mathbf{c}_0$.
(2) Sample a collision-free run length $\ell \sim \mathbf{coll}(n, 0, \mathrm{ord}(\mathcal{C}'), \mathrm{gen}(\mathcal{C}'))$, as described in Lemma 4.4.6. If $\ell \geq \ell_{\max}$, set $\ell = \ell_{\max}$, set $\ell_{\mathrm{out}} = \ell$ and `do_collision` = False; otherwise if $\ell < \ell_{\max}$, set $\ell_{\mathrm{out}} = \ell + 1$ and `do_collision` = True.
(3) Let $q = |\Lambda|$. Sample the batch by sampling the values of the $\mathrm{ord}(\mathcal{C}')$-dimensional transition array D by recursively drawing $\Theta(q^{\mathrm{ord}(\mathcal{C}')})$ multivariate hypergeometric samples summing to ℓ, as described in Section 4.4.3.
(4) Execute the batched reactions. For each entry $r \in \mathbb{N}$ of D corresponding to reactants $\mathbf{r}$:
    Draw a sample from a multinomial distribution on r trials, with probabilities proportional to the rate constants of each reaction in $\mathcal{C}'$ having reactant vector $\mathbf{r}$. For each sampled value, apply the result of executing the corresponding reaction that many times, removing the reactants from $\mathbf{c}$ and adding the products to $\mathbf{c}'$. If the corresponding reaction is not passive, also increase `steps` by the sampled value.
(5) If `do_collision` = True, simulate a collision: Draw an ordered list of reactants $\mathbf{r}$ uniformly from $\mathbf{c} + \mathbf{c}'$, conditioned on at least one of them being from $\mathbf{c}'$ (see proof of Lemma 4.4.12 for how). Then, execute a single reaction from $\mathcal{C}'$ with reactant vector $\mathbf{r}$ as described in step 4, and if it is not passive, increment `steps`.
(6) Set $\mathbf{c} := \mathbf{c} + \mathbf{c}'$ (and output it).

Lemma 4.4.8. *On input $\mathcal{C}'$, $\mathbf{c}_0$, and $\ell_{\max}$, and for any volume $v \in \mathbb{R}_{>0}$, the output of Algorithm 4 is distributed as $\mathcal{F}^{\mathrm{dis}}_{\mathcal{C}',\ell_{\max}}(v, \mathbf{c}_0)$.*

Algorithm 5 Discrete-time full simulation

Input: CRN $\mathcal{C} = (\Lambda, R)$, volume $v \in \mathbb{R}_{>0}$, configuration $\mathbf{c}_0$ of $\mathcal{C}$, number of steps $\ell_{\max} \in \mathbb{N}$
Output: Configuration $\mathbf{c}$ of $\mathcal{C}$ distributed as in Corollary 4.4.9

Set $\mathbf{c} = \mathbf{c}_0$ and $\texttt{steps} = 0$. Repeat until $\texttt{steps} = \ell_{\max}$:

(1) Set $k_0 = \|\mathbf{c}\|$, and let $\mathcal{C}'$ be the output of Algorithm 2 on inputs $\mathcal{C}$, v, and k_0. Let $\mathbf{c}' = \mathbf{c} + k_0 \cdot K$.

(2) Call Algorithm 4 on input $\mathcal{C}'$, $\mathbf{c}'$, and $\ell_{\max} - \texttt{steps}$. Set $\mathbf{c}$ to the returned configuration, with all W and K removed. Add the returned number of steps ℓ_{out} to $\texttt{steps}$.

PROOF. Note first that v has no effect on this distribution, as $\mathcal{C}'$ is uniform, so the value of v affects all propensities identically. We couple Algorithm 4 with Algorithm 3, which has the correct distribution by Corollary 4.4.1.

The only difference between the two processes is that Algorithm 4 batches reactions. Thus, we can couple the processes by running Algorithm 3 while keeping track of individual molecules, and whenever a collision occurs (i.e., when a molecule produced since the last collision is chosen as a reactant), using that number of reactions to sample the length of a collision-free run in Algorithm 4. From the perspective of Algorithm 4, nothing is changed by this coupling, because it samples from the exact distribution of collision-free run lengths given in Lemma 4.4.5. If it is possible for this collision-free run length to be greater than $\ell_{\max} - \texttt{steps}$, then whenever this many reactions occur with no collision in Algorithm 3, we also run Algorithm 5 on a batch of this size (with no collision). This maintains the coupling in all cases. □

Together with Lemma 4.3.3, this implies the following:

Corollary 4.4.9. *On input $\mathcal{C}$, v, $\mathbf{c}_0$, and $\ell_{\max}$, the output of Algorithm 5 is distributed as $\mathcal{F}^{\text{dis}}_{\mathcal{C},\ell_{\max}}(v, \mathbf{c}_0)$.*

We now turn to the issue of efficiency. At its core, our algorithm's efficiency comes from batching, allowing the simulation of $\Theta(\sqrt{n})$ reactions in $O(\log n)$ time. Most batching steps will be able to run this many reactions, so long as there are at least $\Theta(\sqrt{n})$ left to simulate.

Lemma 4.4.10. *Suppose that Algorithm 5 calls Algorithm 4 on inputs $\mathcal{C}'$, $\mathbf{c}'$, and $\ell_{\max} - \texttt{steps}$ with $\|\mathbf{c}'\| = n$ and $(\ell_{\max} - \texttt{steps}) \in \Omega(\sqrt{n})$. Then the number of steps returned by Algorithm 4 is $\Omega\left(\frac{\sqrt{n}}{S_{\mathcal{C},v}(\mathbf{c})}\right)$ in expectation (i.e., the iteration simulates this many reactions from the original CRN).*

PROOF. By Lemma 4.4.7, after step 2, the value ℓ, the number of (possibly passive) reactions in the batch, is $\Theta(\sqrt{n})$ in expectation. As argued in Corollary 4.4.9, Algorithm 5 can be coupled

with Algorithm 3. Therefore, we can apply Lemma 4.4.2 to see that each individual reaction in this batch of $\Theta(\sqrt{n})$ reactions has probability $\frac{1}{S_{\mathcal{C},v}(\mathbf{c})}$ of being non-passive. □

The next lemma refers to the other case from Lemma 4.4.10, which is that we only have $O(\sqrt{n})$ steps left before reaching $\ell_{\max}$.

Lemma 4.4.11. *Suppose that Algorithm 5 calls Algorithm 4 on inputs $\mathcal{C}'$, $\mathbf{c}'$, and $\ell_{\max} - \texttt{steps}$ with $(\ell_{\max} - \texttt{steps}) \in O(\sqrt{n})$, and that every configuration $\mathbf{c}$ appearing at the start of an iteration from this point satisfies $S_{\mathcal{C},v}(\mathbf{c}) \leq s$ for some $s \in \mathbb{R}_{>0}$. Then the algorithm will halt in $O(s \log n)$ more iterations in expectation.*

Proof. If $\ell_{\max} - \texttt{steps}$ is in $O(\sqrt{n})$, then most batches will simulate $\ell_{\max} - \texttt{steps}$ reactions. On average, the fraction of these that are non-passive will be $\frac{1}{S_{\mathcal{C},v}(\mathbf{c})}$. We can view the number of iterations to completion from this point as the maximum of $\ell_{\max} - \texttt{steps}$ independently distributed geometric variables with success probability at least s, because each call to Algorithm 4 has independent probability at least s to simulate a non-passive reaction for each reaction it simulates. The given asymptotic expression is a bound on the expectation of this maximum. □

Lemma 4.4.12. *Suppose Algorithm 5 is run on input $\mathcal{C}$, v, $\mathbf{c}_0$ and $\ell_{\max}$. Suppose that every configuration $\mathbf{c}$ appearing at the start of an iteration of this execution satisfies $n_{\min} \leq \|\mathbf{c}\| \leq n_{\max}$ and $S_{\mathcal{C},v}(\mathbf{c}) \leq s$ for some $n \in \mathbb{N}$, $s \in \mathbb{R}_{>0}$. Then the algorithm runs in time $O\left(\frac{q^{\mathrm{ord}(\mathcal{C})} s \ell_{\max} \log(n_{max})}{\sqrt{n_{min}}}\right)$.*

Proof. Step 1 consists of simple transformations on the CRN, so is not relevant to asymptotic analysis as other steps will take longer. Step 2 takes time $O(\mathrm{ord}(\mathcal{C}) \log(n_{\max}))$ by Lemma 4.4.6. Step 3 can be done in time $\Theta(q^{\mathrm{ord}(\mathcal{C})})$, as each multivariate hypergeometric sample can be drawn in constant time [60]. Step 4 also takes time $\Theta(q^{\mathrm{ord}(\mathcal{C})})$, as there are this many entries in D and relevant samples and configuration updates can be done in constant time per entry. Steps 5 and 6 are not costly; the conditional sampling in step 5 can be done by simple combinatorial calculations.

Thus, each iteration takes time $\Theta(q^{\mathrm{ord}(\mathcal{C})})$. By Lemma 4.4.10, it will take on average $O\left(\frac{\ell_{\max}}{s\sqrt{n}}\right)$ iterations until there are $O(\sqrt{n})$ remaining reactions to simulate. By Lemma 4.4.11, the remainder of the algorithm from there does not take too long. □

4.5. Simulating with continuous time

In this section, we show our main theorem:

THEOREM 4.5.1. *Let $\mathcal{C} = (\Lambda, R)$ be a CRN with $|\Lambda| = q$, $\mathbf{c}$ be a configuration of $\mathcal{C}$, and $v \in \mathbb{R}_{>0}$ be volume. Let $p \in (0, \frac{1}{2}]$ be a parameter. Then it is possible to exactly sample from $\mathcal{F}^{\text{con}}_{\mathcal{C},v}(\mathbf{c}, t)$ (that is, to sample the configuration of $\mathcal{C}$ at time t in volume v, starting from $\mathbf{c}$) in time*

$$O\left(\frac{(2q)^{\text{ord}(\mathcal{C})} s\ell \log(n_{\max})}{n_{\min}^{p}} + n_{\max}^{2p/3}\ell^{1/3}\log(n_{\max})\right),$$

so long as all ℓ configurations $\mathbf{c}$ occurring up to time t satisfy $n_{\min} \leq \|\mathbf{c}\| \leq n_{\max}$ and $S_{\mathcal{C},v}(\mathbf{c}) \leq s$ (see Definition 4.2.8 for the definition of $S_{\mathcal{C},v}(\mathbf{c})$).

This theorem is directly implied by Lemma 4.5.2 and Lemma 4.5.3. See Section 4.5.3 for a discussion of the asymptotics.

So far, we have described an algorithm that exactly simulates CRNs in *discrete time*. That is, it can sample accurately from the distribution of *executions* of a CRN from a given configuration, and then output some subsequence of that execution (outputting the entire execution would take linear time). We also wish to know the (continuous) *inter-reaction times* - that is, not just what configuration the CRN is in and how many reactions have occurred (i.e., "discrete time"), but also how long it took to get there: a *timestamp*. Sampling this information efficiently is surprisingly difficult compared to merely sampling discrete time information. Our approach is to exploit the inert W molecule introduced in Algorithm 1. By carefully modifying the count of this molecule between batches, we ensure that the batching algorithm repeatedly cycles through the same small ($\Theta(n^p)$ for some $p \in (0, \frac{1}{2}]$) set of distinct molecular counts. This allows us to take advantage of a method called adaptive rejection sampling [**29**] to quickly sample inter-reaction times from the same distribution repeatedly, rather than sampling the time of each individual reaction.

Throughout this section, let $\mathbf{c} \in \mathbb{N}^{\Lambda}$ be a configuration of the CRN $\mathcal{C}$ at the start of a batch, let $n = \|\mathbf{c}\|$, let $k \in \mathbb{N}^{+}$ denote the number of interactions for which we wish to sample the total inter-reaction time, let $o = \text{ord}(\mathcal{C})$, and let $g = \text{gen}(\mathcal{C})$. Like with individual reactions, sampling individual inter-reaction times would take linear time, so we must sample the sum of many inter-reaction times together. Each individual inter-reaction time is distributed as an exponential random variable with rate equal to total propensity $p^{\text{tot}}_{\mathbf{c},v}$. For uniformly reactive CRNs this value is always equal to a constant times $\binom{n}{o}$. Since each reaction increases the total molecular count by g, the i'th reaction has a rate proportional to $\binom{n+ig}{o}$; it follows that for a batch of size k, we must sample the

variable

$$\mathbf{H} = \sum_{i=0}^{k-1} \mathbf{X}\left(\binom{n+ig}{o}\right),$$

where each $\mathbf{X}(\lambda)$ is independently exponentially distributed with rate λ. The distribution of such a variable $\mathbf{H}$ is known as a *hypoexponential distribution*.[7] Our aim here is to sample from it efficiently.

We first show (Sections 4.5.1 and 4.5.2) a theoretical approach allowing us to efficiently sample from $\mathbf{H}$ *exactly* under an assumption of bounded molecular count. We then discuss (Section 4.5.4) more practical approaches for sampling approximately from $\mathbf{H}$, which are faster and appear in theory and practice to be accurate enough that they do not meaningfully affect the output distribution.

4.5.1. Sampling exactly from the hypoexponential distribution. To drastically simplify this discussion, we assume that the execution we simulate has bounded total molecular count. This assumption is true of any physically realistic CRN execution. It also does not meaningfully deviate from what the Gillespie algorithm can simulate; for example, on CRNs which exhibit finite-time blowup, any algorithm attempting to sample a configuration at some given time must have some probability of failure. Even on CRNs that do not exhibit finite-time blowup, unbounded molecular counts typically occur as a result of exponential growth, a case where the Gillespie algorithm will take exponential time and is not practical.

The PDF P of a hypoexponential distribution with rates $\lambda_1, \ldots, \lambda_k$ is [51]:

$$P(t) = \sum_{i=1}^{k} C_{i,k} \lambda_i e^{-\lambda_i t}, \tag{12}$$

where

$$C_{i,k} = \prod_{j \in \{1,\ldots,k\} \setminus \{i\}} \frac{\lambda_j}{\lambda_j - \lambda_i}.$$

Naïvely computing all $C_{i,k}$ appears to require $\Theta(k^2)$, taking time $O(k)$ for each of the k products $C_{i,k}$. (Each binomial coefficient $\lambda_i = \binom{n+ig}{o}$ can be computed in $O(1)$ time since we consider $o, g = O(1)$ with respect to n.) However, we can improve this to $\Theta(k \log^2 k)$ time. First note that

[7] In the case where the CRN is conservative, that is, each reaction has an equal number of reactants and products, the timestamps follow an Erlang distribution, i.e., a sum of independent and *identically* distributed exponential random variables.

we can write

$$C_{i,k} = \prod_{j \in \{1,\ldots,k\} \setminus \{i\}} \frac{\lambda_j}{\lambda_j - \lambda_i} = \frac{\prod_{j \neq i} \lambda_j}{\prod_{j \neq i} (\lambda_j - \lambda_i)}.$$

The top product can be handled in this way. We use time $\Theta(k)$ to compute the full product $A = \prod_{j=1}^{k} \lambda_j$. We then compute each term $\prod_{j \neq i} \lambda_j$ for $1 \leq i \leq k$ in time $O(1)$ by a single division A/λ_i. Thus the top products can be computed in total time $\Theta(k)$.

Now consider the bottom products $\prod_{j \neq i} (\lambda_j - \lambda_i)$. Consider the polynomial $f(x) = \prod_{j=1}^{k} (\lambda_j - x)$. Let $f'(x) = \frac{d}{dx} f(x)$. Then by the product rule,

$$f'(x) = \sum_{i=1}^{k} \frac{d}{dx}(\lambda_i - x) \cdot \left(\prod_{j \neq i} (\lambda_j - x) \right) = -\sum_{i=1}^{k} \left(\prod_{j \neq i} (\lambda_j - x) \right).$$

If we evaluate $f'(\lambda_m)$ for $1 \leq m \leq k$, for terms $i \neq m$ in the sum, one factor $(\lambda_j - x)$ in the product is 0 (specifically for $j = m$, so those terms of the sum vanish), and we have $C_{i,k} = \prod_{j \neq i} (\lambda_j - \lambda_i) = -f'(\lambda_i)$. Thus it suffices to construct f and evaluate its derivative at λ_i to compute $C_{i,k}$.

To evaluate at these points efficiently (time $O(k \log^2 k)$ for all k products), the first step is to convert the factored polynomial $f(x) = \prod_{j=1}^{k} (\lambda_j - x)$ into its expanded coefficient form. This can be done with a standard divide-and-conquer recursion, splitting into two polynomials with $k/2$ factors each. The base case is to FOIL the degree-2 $(\lambda_i - x)(\lambda_{i+1} - x) = \lambda_i \lambda_{i+1} - (\lambda_i + \lambda_{i+1})x + x^2$. The recursive case can be handled using standard FFT-based polynomial multiplication routines [20], taking time $\Theta(k \log k)$ to multiply two degree-k polynomials. Since this will have $\log k$ levels of recursion and spend time $O(k \log k)$ at each level of recursion, the total time required is $O(k \log^2 k)$. Now that we have computed the polynomial $f'(x)$, similar FFT-based methods for *multipoint evaluation* [62] can be used to evaluate f' at k points in time $O(k \log^2 k)$. (More generally time $O(k + m) \log^2 (k + m)$ to evaluate a degree-k polynomial at m points; $m = k$ in our case.) Thus in time $O(k \log^2 k)$, we can compute the coefficients $C_{i,k}$ used in the definition of the PDF and CDF; so long as we continue sampling from the same hypoexponential; these values do not need to be recomputed, reducing the amortized cost the more reactions are executed in the total simulation.

Nevertheless, once we have the coefficients $C_{i,k}$, it still requires time $\Theta(k)$ to evaluate the PDF in (12). However, to achieve an asymptotic speedup over the Gillespie algorithm, we must sample a hypoexponential defined by k exponentials in time asymptotically smaller than k, so time $\Theta(k)$ remains too expensive. However, this PDF is log-concave: that is, the second derivative of $\log(P(t))$

is negative for all $t > 0$.[8] This allows us to use the black-box method of adaptive rejection sampling, outlined in [29], to sample repeatedly from the distribution without having to evaluate the PDF every time, as with normal rejection sampling. In short, this method works by establishing upper and lower piecewise linear bounds on $\log(P(t))$, and improving these bounds to match the function closely as more evaluations are made. The more accurate these bounds are, the less likely that drawing a sample requires one to explicitly compute the PDF. In [66], the authors give empirical evidence and outline a proof that this method generally allows one to obtain $O(m)$ samples from such a distribution in only $O(\sqrt[3]{m})$ evaluations of the PDF. We will assume that this is the case.

In order to take advantage of this, we must guarantee that we sample from the same hypoexponential distribution repeatedly. The key insight is that W can be freely added or removed without altering any propensities corresponding to the original CRN, since W is inert, but its presence changes what hypoexponential distribution will be sampled. If we repeatedly execute some number of (possibly passive) reactions r, then remove $r \cdot g$ copies of W from the configuration (i.e., return to the molecular count before the r reactions were executed), then each such cycle will have total elapsed times that are distributed like independent identical hypoexponentials. Because we assume an upper bound $n_{\max}$ on molecular count, we can guarantee there will always be enough W to remove. If no such bound exists, our algorithm remains correct, but may be inefficient. To prevent these extra W from causing too many passive reactions, we can switch to a different hypoexponential whenever the molecular count halves (or doubles up to $n_{\max}$). This adds an extra $\log n_{\max}$ factor to the analysis.

4.5.2. Rejection sampling for exact end times. The typical input to the Gillespie algorithm contains an exact continuous time t at which to sample a configuration. We will eventually run past time t during a batch, but wish to sample the configuration at time exactly t. To avoid this issue while remaining exact, we use rejection sampling to sample how many reactions from the batch occur before time t, under the assumption that the last reaction finishes after time t.

The simplest version of this procedure is as follows: when the end time of a batch is sampled to be past t, sequentially sample the exponential variables that comprise the batch. The first such sample that would cause simulated time to go past t indicates the first reaction in the batch that

[8]This follows because the hypoexponential distribution is a convolution of exponentials, which are log-concave, and convolution preserves log-concavity.

happens *after* time t, so we simulate everything before it, without a collision (as the collision would happen after time t). If all times are sampled and t still has not been reached, we reject the sample and start over. If the probability in some batch step exceeding time t is p, then the probability of the sample being accepted is also p, leading to on average $\frac{1}{p}$ rejections. Therefore, each time a batch step is run, the expected number of rejections is 1.

Generally, it is fine to run this rejection sampling in this way, as sampling a single batch slowly does not affect asymptotics. If some low-probability event causes a significant number of rejections, we can instead sample directly from the conditional distribution of the hypoexponential in question. This is expensive, but only necessarily with probability that is controllably low.

This can be made more efficient via binary searching by sampling from hypoexponential distributions. However, on any input t on which we expect $\Omega(n)$ reactions to happen, this is not necessary, as the last batch contains $\Theta(\sqrt{n})$ reactions in expectation (see Lemma 4.4.7, so sampling the last batch slowly does not affect asymptotics.

4.5.3. Full continuous-time simulation algorithm. Here we provide our main algorithm, Algorithm 6. Like Algorithm 5, it works by repeatedly calling Algorithm 4. However, it does so in such a way that allows repeated sampling from the same hypoexponential distribution. Note in particular that for all of these algorithms, we describe the output as a single configuration, but in practice we would sample a sequence of configurations by calling such algorithms repeatedly.

Algorithm 6 Continuous-time exact simulation

Input: CRN $\mathcal{C} = (\Lambda, R)$, volume $v \in \mathbb{R}_{>0}$, configuration $\mathbf{c}_0$ of $\mathcal{C}$, end time $t_{\max}$, batching parameter $p \in \left(0, \frac{1}{2}\right]$

Output: Configuration $\mathbf{c}$ of $\mathcal{C}$ distributed as in Lemma 4.5.2

Set $\mathbf{c} = \mathbf{c}_0$ and $t = 0$. Repeat until step 2 exits:

(1) Let i be the least integer where $\|\mathbf{c}\| \leq 2^i$, $k_0 = \|\mathbf{c}\|$, $n_0 = 2^{i+1}$, $\ell_{\max} = \lfloor \|\mathbf{c}\|^p \rfloor$, and $\mathcal{C}' =$ the output of Algorithm 2 on inputs $\mathcal{C}$, v, and k_0. Let $\mathbf{c}' = \mathbf{c} + k_0 \cdot K + (n_0 - \|\mathbf{c}\| - k_0) \cdot W$.

(2) Sample a value t_0 from the hypoexponential distribution (see Section 4.5.1) with rates $\lambda_i = \binom{n_0 + i \cdot \mathrm{gen}(\mathcal{C})}{\mathrm{ord}(\mathcal{C})}, 0 \leq i < \ell_{\max}$. If $t_0 + t > t_{\max}$, run end-of-simulation rejection sampling starting from $\mathbf{c}'$ to sample a configuration $\mathbf{c}$ (see Section 4.5.2), and then output $\mathbf{c}$ with all K and W removed. Otherwise, add t_0 to t.

(3) Repeat until $\ell_{\max} = 0$: run Algorithm 4 on inputs $\mathcal{C}'$, $\mathbf{c}'$, and $\ell_{\max}$. Subtract the returned number of steps from $\ell_{\max}$. Set $\mathbf{c}'$ to the returned configuration.

(4) Set $\mathbf{c}$ to $\mathbf{c}'$ with all K and W removed.

Lemma 4.5.2. *On input $\mathcal{C}$, $\mathbf{c}$, v, and $t_{\max}$, the output of Algorithm 6 is distributed as $\mathcal{F}^{\mathrm{con}}_{\mathcal{C},v}(\mathbf{c}, t_{\max})$.*

PROOF. We couple Algorithm 6 with the Gillespie algorithm run on $\mathcal{C}'$ over the course of each loop iteration. By Lemma 4.3.3, this is equivalent to the Gillespie algorithm run on $\mathcal{C}$ over distributions of species other than K and W, even in continuous time. At the start of the iteration, both algorithms operate on the same configuration $\mathbf{c}'$ described in step 1. If the Gillespie algorithm simulates $\ell_{\max}$ reactions within time $t_{\max} - t$, Algorithm 6 runs an iteration where step 2 samples such a time. In this case, Lemma 4.4.8 implies that the coupling remains valid. If the Gillespie algorithm runs past time $t_{\max}$ before simulating this many reactions, the coupling remains valid because, by the discussion in Section 4.5.2 Algorithm 6 correctly conditionally samples how many reactions to simulate. □

Lemma 4.5.3. *Suppose Algorithm 6 is run on input $\mathcal{C}$, v, $\mathbf{c}_0$, $t_{\max}$, and p, simulating an execution of $\mathcal{C}$ that has ℓ reactions. Suppose that every configuration $\mathbf{c}$ appearing at the start of an iteration of this execution satisfies $n_{\min} \leq \|\mathbf{c}\| \leq n_{\max}$ and $S_{\mathcal{C},v}(\mathbf{c}) \leq s$ for some $n_{\min}, n_{\max} \in \mathbb{N}$, $s \in \mathbb{R}_{>0}$. Then the algorithm runs in time $O\left(\frac{(2q)^{\mathrm{ord}(\mathcal{C})} s\ell \log(n_{\max})}{n_{\min}^p} + n_{\max}^{2p/3}\ell^{1/3}\log(n_{\max})\right)$.*

To break down this expression: $\mathrm{ord}(\mathcal{C})$ and q are constants depending only on $\mathcal{C}$. s is a slowdown factor due to effects of the specific configuration being simulated. Empirically, for example, $s \leq 5$ for the Lotka-Volterra oscillator, as shown in Figure 5.4. On many reasonable inputs $n_{\min}$ and $n_{\max}$ will differ by a multiplicative constant, so we treat them as being the same, and ignore the logarithmic factors. With these simplifications, we can express the runtime as roughly

$$O\left(n^{-p}\ell + n^{2p/3}\ell^{1/3}\right).$$

The first term is the cost of sampling configurations in discrete time using batching. It decreases as we increase p toward $\frac{1}{2}$, because we are able to run more reactions in each batch. The second term is the cost of adaptive rejection sampling to exactly sample inter-reaction times. It decreases as we decrease p, because this allows us to sample more frequently from a less complex hypoexponential distribution, allowing adaptive rejection sample to learn about the distribution more quickly. Because of this, we can consider regimes comparing ℓ and n, and find the optimal asymptotic runtime of our algorithm in each regime by setting these two exponents equal to each other. If $\ell \in \Omega(n^{5/4})$, the first term is dominant even when $p = \frac{1}{2}$, which is the largest value of p that is beneficial (because there are typically $\Theta(n^{1/2})$ reactions between collisions). In this regime, the simulated

execution is long enough that the adaptive rejection sampling algorithm has enough time to learn the hypoexponential distribution, and there is no asymptotic slowdown compared to our discrete time algorithm. If $\ell \in \Theta(n)$, the asymptotically optimal value of p is $2/5$, and our algorithm gives a speedup factor over the Gillespie algorithm of $n^{2/5}$. We generally expect $\ell \in \Omega(n)$, because CRNs do not generally exhibit interesting behavior in a sublinear number of reactions.

PROOF. The first term comes from the runtime of step 3, which is shown in Lemma 4.4.12. There are two relevant differences: first, $\sqrt{n}$ is replaced by $n^p_{\min}$, because this factor comes from the number of reactions that are batched, and Algorithm 6 only calls Algorithm 4 to simulate n^p reactions at a time. Second, up to half of the molecules in $\mathbf{c}'$ might be W. This might cause passive reactions to be simulated more often, but each simulated reaction contains no W in its reactants with probability $\Omega(2^{-\operatorname{ord}(\mathcal{C})})$, which is combined with the $q^{\operatorname{ord}(\mathcal{C})}$ term. The only other relevant cost in Algorithm 6 is step 2, which samples the hypoexponential distribution. To simulate ℓ reactions, Algorithm 6 must sample a hypoexponential distribution with n^p rates (representing the time to run n^p reactions) a total of $\frac{\ell}{n^p}$ times. We use adaptive rejection sampling, which allows us to obtain these $\frac{\ell}{n^p}$ samples in $\sqrt[3]{\frac{\ell}{n^p}}$ evaluations of the hypoexponential PDF. To evaluate the hypoexponential PDF for the first time, we must first compute the values C_{i,n^p} given in Eq. (12), which can be done in time $O(n^p)$ as shown in Section 4.5.1. Then, each subsequent evaluation takes time $O(n^p)$ to compute and sum n^p terms. It follows that this process takes time $n^p \cdot \sqrt[3]{\frac{\ell}{n^p}}$ for a given hypoexponential distribution. The process may need to sample from $\log(n_{\max})$ such distributions as molecular count changes. □

4.5.4. Sampling approximately from the hypoexponential distribution. Implementations of the algorithm described in Sections 4.5.1 and 4.5.2 spend significant time sampling from the hypoexponential distribution. Although the asymptotic performance is adequate amortized over many calls to the sampling procedure, the constants involved make this a performance bottleneck in practice.

In this section we describe an alternative that is faster in practice and produces nearly identical outcomes. We give up trying to sample from the hypoexponential distribution exactly, instead sampling from the computationally much simpler *gamma* distribution. This technically means that in practice we are not sampling from precisely the same distribution of times as the Gillespie

algorithm. We emphasize that this is merely a slight imprecision in *timestamps*; the sequence of configurations is still sampled from precisely the same distribution as Gillespie. Thus this will not lead to inaccuracies in the sampled qualitative behavior of the CRN as with inexact methods such as τ-leaping. Furthermore, we justify that the measured deviations of timestamps from those of Gillespie will be negligible, i.e., in any reasonably large population, so small that it would not change a single pixel on a plot of counts over time. Compared to τ-leaping, which gives worse approximations with larger τ, we have no time-accuracy tradeoff. There are just some places where we observed a prohibitive computational cost in practice, despite the asymptotic performance shown in Section 4.5.1. The optimizations described in this section are therefore largely relevant to practical implementation, so we focus on whether the approximation leads to inaccurate results in practice.

The approximations described in this section actually get better for larger population sizes n. Thus in practice, for small n one can sample the exact time distribution directly, using the approximations described in this section only on n sufficiently large that the approximation is so accurate that deviations from the true distribution are undetectable in practice.

Recall that the special case of the hypoexponential, where each of the $k \in \mathbb{N}^+$ exponentials being summed has identical rate λ, is called an *Erlang* distribution $\mathbf{Er}(k, \lambda)$. The gamma distribution $\mathbf{\Gamma}(\alpha, \lambda)$ generalizes $\mathbf{Er}(k, \lambda)$ to allow a real-valued first parameter α, but coincides with $\mathbf{Er}(\alpha, \lambda)$ for positive integer $\alpha \in \mathbb{N}^+$. The PDF of the Erlang has a term $(k-1)!$, and the gamma function $\Gamma : \mathbb{C} \to \mathbb{C}$, defined on all complex numbers, has the property that $\Gamma(k) = (k-1)!$ for all positive integers k. The gamma distribution has the same PDF as the Erlang, but with $\Gamma(k)$ appearing in place of the $(k-1)!$ factorial in Erlang's PDF.

The reason we use the gamma distribution instead of the Erlang to approximate the hypoexponential distribution is that we use the method of *moment matching* [10], finding the gamma distribution with the same mean (first moment) and variance (second moment) as the desired hypoexponential. Furthermore, the rates of the exponentials defining the hypoexponential are very close to each other: since $k = \Theta(\sqrt{n})$ in expectation, the ratio of the first and last terms of the sum defining the mean is very close to 1. If the ratio were *equal* to 1, then this would be a simple Erlang distribution. The hypoexponential has many real parameters, but Erlang only has 2, and the first is an integer. Thus with the integer restriction, we cannot hope to find an Erlang matching both of

these moments precisely. Yet the *gamma* distribution can match both, since its two parameters are both real-valued, i.e., the gamma has the same number of degrees of freedom as the two moments we want to match. Empirically in practice, the real-valued *shape* parameter of the gamma is very close to an integer, so is "almost" an Erlang.

We note that it is possible to have a controllable approximation scheme that trades accuracy for speed. The hypoexponential is defined as a sum of k exponential random variables. We could have a parameter $1 \leq c \leq k$ and sample c gamma random variables, with the i'th gamma having expected value to match the i'th block of k/c exponentials. In the case of $c = 1$, this is the approximation we described above. In the case of $c = k$, this samples exactly the hypoexponential by individually sampling the k exponentials defining it. In between, as c is larger, this is a better and better approximation. However, in practice, we found that setting $c = 1$ leads to a distribution essentially indistinguishable from the hypoexponential being approximating.

The hypoexponential in our case is parameterized by four parameters:

- n: population size
- k: number of interactions in a collision-free run
- o: order of the CRN (number of reactants)
- g: generativity of the CRN (number of products minus number of reactants)

Recall that the exponential random variables of the inter-reaction times have rates $\binom{n}{o}$, $\binom{n+g}{o}$, $\binom{n+2g}{o}$, $\ldots$, $\binom{n+(k-1)g}{o}$. Those are the rates for the hypoexponential distribution giving the time of the entire batch of length k. Such a distribution has

$$\text{mean} \qquad \mu = \sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}} \qquad \text{and variance} \qquad \sigma^2 = \sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}^2}.$$

Computing these directly would take time $\Omega(k)$, defeating the goal of processing a batch of size k in time $o(k)$. (Though in practice we do compute them directly for small values of n.) The rest of Section 4.5.4 is devoted to showing that we can more efficiently compute the mean and variance of this hypoexponential distribution.

4.5.4.1. *Technical lemmas.* We first prove several technical lemmas involving identities that will be useful for computing both the mean and variance of a hypoexponential with the rates relevant to our batching algorithm.

Lemma 4.5.4. *For all $B, o \in \mathbb{N}^+$ such that $o \leq B$,*

$$\frac{1}{\binom{B}{o}} = o \cdot \sum_{m=0}^{o-1} \binom{o-1}{m} \frac{(-1)^m}{B-(o-1-m)}.$$

Proof. First,

$$\frac{1}{\binom{B}{o}} = \frac{o!(B-o)!}{B!} = o \cdot \frac{(o-1)!}{\prod_{j=0}^{o-1} B-j}$$

so it suffices to show

$$\frac{(o-1)!}{\prod_{j=0}^{o-1} B-j} = \sum_{m=0}^{o-1} \binom{o-1}{m} \frac{(-1)^m}{(B-(o-1-m))}. \tag{13}$$

Decomposing by partial fractions we can write:

$$\frac{1}{\prod_{j=0}^{o-1} B-j} = \sum_{j=0}^{o-1} \frac{A_j}{B-j}. \tag{14}$$

where the coefficients A_j are given by the residue formula:

$$A_j = \frac{1}{\prod_{\ell=0,\ell\neq j}^{o-1}(j-\ell)} = \frac{1}{\prod_{\ell=0}^{j-1}(j-\ell)\cdot\prod_{\ell=j+1}^{o-1}(j-\ell)} = \frac{1}{j!\prod_{\ell=j+1}^{o-1}(j-\ell)}.$$

For the product $\prod_{\ell=j+1}^{o-1}(j-\ell)$, when ℓ takes values $j+1, j+2, \ldots, o-1$, then $(j-\ell)$ takes values $-1, -2, \ldots, -(o-1-j)$. Conventionally factorial is not defined on negative integer values, but this product's absolute value is $(o-j-1)!$, equal to $(o-1-j)!$ if $o-1-j$ is even or $-((o-1-j)!)$ if $o-1-j$ is odd. Written differently, $\prod_{\ell=j+1}^{o-1}(j-\ell) = (-1)^{o-1-j}(o-1-j)!$. Thus we have

$$A_j = \frac{1}{j!(-1)^{o-1-j}(o-1-j)!} = \frac{(-1)^{o-1-j}}{j!(o-1-j)!}.$$

Substituting into Equation (14),

$$\frac{1}{\prod_{j=0}^{o-1}(B-j)} = \sum_{j=0}^{o-1} \frac{(-1)^{o-1-j}}{j!(o-1-j)!(B-j)} = \sum_{m=0}^{o-1} \frac{(-1)^m}{m!(o-1-m)!(B-(o-1-m))},$$

where the second equality follows by letting $m = o-1-j$ (i.e., add the terms of the sum in the reverse order). Now multiply both sides by $(o-1)!$ to show Equation (13) holds:

$$\frac{(o-1)!}{\prod_{j=0}^{o-1}(B-j)} = \sum_{m=0}^{o-1} \frac{(o-1)!(-1)^m}{m!(o-1-m)!(B-(o-1-m))} = \sum_{m=0}^{o-1} \binom{o-1}{m} \frac{(-1)^m}{B-(o-1-m)}. \qquad \square$$

For each $n \in \mathbb{N}$, we let $\psi_n : \mathbb{R} \to \mathbb{R}$ denote the *n'th polygamma* function with real arguments, the $(n+1)$'st derivative of the "log gamma" function $\ln \Gamma(x)$. In particular, $\psi_0(x) = (\ln \Gamma(x))' = \frac{\Gamma(x)'}{\Gamma(x)}$ and $\psi_1 = (\ln \Gamma(x))''$ are respectively known as the *digamma* and *trigamma* functions, where $f(x)'$ and $f(x)''$ respectively denote $\frac{df(x)}{dx}$ and $\frac{d^2 f(x)}{dx^2}$.

The following is a technical lemma relating sums of a certain form to differences in the digamma function ψ_0.

Lemma 4.5.5. *For all $A, k, g \in \mathbb{N}^+$,*

$$\sum_{i=0}^{k-1} \frac{g}{A+ig} = \psi_0\left(k + \frac{A}{g}\right) - \psi_0\left(\frac{A}{g}\right).$$

PROOF. We use the identity [5, 49] for $k \in \mathbb{N}^+$,

$$\psi_0(k+z) = \sum_{i=0}^{k-1} \frac{1}{z+i} + \psi_0(z). \tag{15}$$

Then

$$\begin{aligned}\sum_{i=0}^{k-1} \frac{g}{A+ig} = \sum_{i=0}^{k-1} \frac{1}{\frac{A+ig}{g}} = \sum_{i=0}^{k-1} \frac{1}{\frac{A}{g}+i} &= \sum_{i=0}^{k-1} \frac{1}{\frac{A}{g}+i} + \psi_0\left(\frac{A}{g}\right) - \psi_0\left(\frac{A}{g}\right) \\ &= \psi_0\left(k + \frac{A}{g}\right) - \psi_0\left(\frac{A}{g}\right) \qquad \text{by Equation (15).}\end{aligned}$$

□

The following similar technical lemma involves squaring the terms in the sum of Lemma 4.5.5, which turns out to be the difference of two *trigamma* functions ψ_1.

Lemma 4.5.6. *For all $A, k, g \in \mathbb{N}^+$,*

$$\sum_{i=0}^{k-1} \left(\frac{g}{A+ig}\right)^2 = \psi_1\left(\frac{A}{g}\right) - \psi_1\left(k + \frac{A}{g}\right)$$

PROOF. The trigamma function ψ_1 has infinite series expansion [42] $\psi_1(z) = \sum_{i=0}^{\infty} \frac{1}{(z+i)^2}$, so

$$\psi_1\left(\frac{A}{g}\right) = \sum_{i=0}^{\infty} \frac{1}{\left(\frac{A}{g}+i\right)^2} = \sum_{i=0}^{\infty} \frac{1}{\left(\frac{A+ig}{g}\right)^2} = \sum_{i=0}^{\infty} \left(\frac{g}{A+ig}\right)^2$$

and similarly

$$\psi_1\left(k+\frac{A}{g}\right)=\sum_{i=0}^{\infty}\frac{1}{\left(\frac{A}{g}+k+i\right)^2}=\sum_{i=0}^{\infty}\left(\frac{g}{(A+(k+i)g)}\right)^2=\sum_{m=k}^{\infty}\left(\frac{g}{A+mg}\right)^2,$$

where the last re-indexes the sum letting $m=k+i$. Now observe

$$\psi_1\left(\frac{A}{g}\right)-\psi_1\left(k+\frac{A}{g}\right)=\sum_{i=0}^{\infty}\left(\frac{g}{A+ig}\right)^2-\sum_{m=k}^{\infty}\left(\frac{g}{A+mg}\right)^2=\sum_{i=0}^{k-1}\left(\frac{g}{A+ig}\right)^2. \qquad \square$$

Lemma 4.5.7. *For all $A,B,k,g\in\mathbb{N}^+$ with $A\neq B$,*

$$\sum_{i=0}^{k-1}\frac{1}{(A+ig)(B+ig)}=\frac{1}{g(A-B)}\cdot\left[\psi_0\left(\frac{A}{g}\right)-\psi_0\left(k+\frac{A}{g}\right)-\psi_0\left(\frac{B}{g}\right)+\psi_0\left(k+\frac{B}{g}\right)\right].$$

Proof. Note that by partial fraction decomposition,

$$\frac{1}{B+ig}-\frac{1}{A+ig}=\frac{A+ig-(B+ig)}{(A+ig)(B+ig)}=\frac{A-B}{(A+ig)(B+ig)},$$

so dividing both sides by $A-B$:

$$\frac{1}{(A+ig)(B+ig)}=\frac{1}{A-B}\left(\frac{1}{B+ig}-\frac{1}{A+ig}\right).$$

Thus

$$\begin{aligned}\sum_{i=0}^{k-1}\frac{1}{(A+ig)(B+ig)}&=\sum_{i=0}^{k-1}\frac{1}{A-B}\left(\frac{1}{B+ig}-\frac{1}{A+ig}\right)\\&=\frac{1}{A-B}\cdot\left[\sum_{i=0}^{k-1}\frac{1}{B+ig}-\sum_{i=0}^{k-1}\frac{1}{A+ig}\right]\\&=\frac{1}{g(A-B)}\cdot\left[\sum_{i=0}^{k-1}\frac{g}{B+ig}-\sum_{i=0}^{k-1}\frac{g}{A+ig}\right]\\&=\frac{1}{g(A-B)}\cdot\left[\psi_0\left(k+\frac{B}{g}\right)-\psi_0\left(\frac{B}{g}\right)-\psi_0\left(k+\frac{A}{g}\right)+\psi_0\left(\frac{A}{g}\right)\right],\end{aligned}$$

where the final equality follows by applying Lemma 4.5.5 to each sum. The lemma follows by rearranging terms in the brackets. $\square$

4.5.4.2. *Computing mean and variance of hypoexponentials.* The mean of a hypoexponential distribution defined as a sum of exponential random variables with rates $\binom{n}{o}$, $\binom{n+g}{o}$, $\binom{n+2g}{o}$, …, $\binom{n+(k-1)g}{o}$, i.e., with means equal to the reciprocals of those rates, by linearity of expectation, is

$\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}}$. Calculating this directly by computing the sum takes time $\Omega(k)$, but we need to do this when processing a batch of size k, and the entire point of the algorithm is to use much less than time k to process k reactions.

The next lemma allows us to compute the mean of a such a hypoexponential with k terms by computing a sum with only o terms. This is significant because we'll think of o (the maximum number of reactants in any reaction in the original CRN) as a small constant, whereas the naïve way to calculate the mean (the left side of the equation of Lemma 4.5.8) would require time $\Theta(k)$, where $k \gg o$, to sum all expected values $\frac{1}{\binom{n+ig}{o}}$. The identities proven in Section 4.5.4.1 involving the digamma ψ_0 and trigamma ψ_1 functions will be used; fortunately, algorithms exist to compute these functions in time $O(1)$ [59].[9]

We note that in actual implementation, these identities are extremely sensitive to floating-point rounding errors. In particular, even if each term t in the sum of, e.g., Lemma 4.5.8 is "moderately sized", i.e., within a few orders of magnitude of 1, in general there are pairs of terms, e.g., $10^{-2} \leq t_1, t_2 \leq 10^2$, such that $|t_1 - t_2| \ll 10^{-15}$, i.e., the terms suffer "catastrophic cancellation" in which pairwise differences are much smaller than the precision of standard floating-point arithmetic.[10] Thus, when computing these sums, it is necessary to use arbitrary precision libraries such as Python's `mpmath` package [1] to avoid such catastrophic cancellation errors.

Lemma 4.5.8. *For all $n, k, o, g \in \mathbb{N}^+$, where $o \leq n$,*

$$\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}} = \frac{o}{g} \cdot \sum_{m=0}^{o-1} (-1)^m \cdot \binom{o-1}{m} \cdot \left[\psi_0 \left(k + \frac{n-(o-1-m)}{g} \right) - \psi_0 \left(\frac{n-(o-1-m)}{g} \right) \right].$$

[9] More precisely in time depending only on the desired relative error, but independent of the magnitude of the argument. See also [1, 8, 54].

[10] Although IEEE 64-bit double-precision floating point numbers can be as small as 10^{-308}, taking the difference of two floats close to 1, since they use about 15 digits of precision, can only represent differences between such numbers as small as 10^{-15}, e.g., $1.000000000000000005 - 1.000000000000000004$ is equal to 0.000000000000000001, yet the above expression evaluated with double-precision floats evaluates to 0.0 since both the float literals 1.000000000000000005 and 1.000000000000000004 evaluate to 1.0. So in evaluating the terms of the sums such as in Lemma 4.5.8, we must take care that the individual terms are evaluated with sufficient precision that two different opposite-sign terms with very close absolute values are not rounded to have identical absolute values.

PROOF. Letting $B = n + ig$ in Lemma 4.5.4, we have

$$\begin{aligned}
\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}} &= \sum_{i=0}^{k-1} o \cdot \sum_{m=0}^{o-1} \binom{o-1}{m} \frac{(-1)^m}{n+ig-(o-1-m)} && \text{by Lemma 4.5.4} \\
&= o \cdot \sum_{m=0}^{o-1} (-1)^m \cdot \binom{o-1}{m} \cdot \sum_{i=0}^{k-1} \frac{1}{n-(o-1-m)+ig} \\
&= \frac{o}{g} \cdot \sum_{m=0}^{o-1} (-1)^m \cdot \binom{o-1}{m} \cdot \sum_{i=0}^{k-1} \frac{g}{n-(o-1-m)+ig} \\
&= \frac{o}{g} \cdot \sum_{m=0}^{o-1} (-1)^m \cdot \binom{o-1}{m} \cdot \left[\psi_0 \left(k + \frac{n-(o-1-m)}{g} \right) - \psi_0 \left(\frac{n-(o-1-m)}{g} \right) \right],
\end{aligned}$$

where the last equality follows from Lemma 4.5.5 with $A = n - (o - 1 - m)$. □

To analyze the time complexity of computing the right-hand side of Lemma 4.5.8, observe that the binomial coefficients can be computed iteratively while evaluating the sum, via the identity $\binom{o-1}{m} = \binom{o-1}{m-1} \cdot \frac{o-m}{m}$, so that the entire sum requires time $O(o)$ to compute, since ψ_0 can be computed in time $O(1)$.

Similarly to the mean, by linearity of variance when the random variables are independent, the variance of a hypoexponential distribution, defined by rates $\binom{n}{o}$, $\binom{n+g}{o}$, $\binom{n+2g}{o}$, …, $\binom{n+(k-1)g}{o}$, is $\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}^2}$. The next lemma, similarly to Lemma 4.5.8, allows the variance to be computed in time $O(o^2)$.

Lemma 4.5.9. *For all $n, k, o, g \in \mathbb{N}^+$, where $o \le n$,*

$$\begin{aligned}
\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}^2} =& \frac{o^2}{g^2} \sum_{m=0}^{o-1} \binom{o-1}{m}^2 \left[\psi_1 \left(\frac{n-(o-1-m)}{g} \right) - \psi_1 \left(k + \frac{n-(o-1-m)}{g} \right) \right] \\
&+ \frac{2o^2}{g} \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} \frac{(-1)^{m+j}}{m-j} \binom{o-1}{m} \binom{o-1}{j} \\
&\qquad \cdot \left[\psi_0 \left(\frac{n-(o-1-m)}{g} \right) - \psi_0 \left(k + \frac{n-(o-1-m)}{g} \right) \right. \\
&\qquad \left. - \psi_0 \left(\frac{n-(o-1-j)}{g} \right) + \psi_0 \left(k + \frac{n-(o-1-j)}{g} \right) \right].
\end{aligned}$$

PROOF. Letting $B = n + ig$ in Lemma 4.5.4, we have

$$\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}^2} = \sum_{i=0}^{k-1} o^2 \cdot \left(\sum_{m=0}^{o-1} \binom{o-1}{m} \frac{(-1)^m}{n+ig-(o-1-m)} \right)^2. \tag{16}$$

Recall the identity[11]

$$\left(\sum_{m=0}^{o-1} x_m\right)^2 = \sum_{m=0}^{o-1} x_m^2 + 2 \cdot \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} x_m \cdot x_j. \tag{17}$$

By Equation (17), we can write the squared sum in Equation (16) as

$$\begin{aligned}
&\left(\sum_{m=0}^{o-1} \binom{o-1}{m} \frac{(-1)^m}{n+ig-(o-1-m)}\right)^2 \\
=&\sum_{m=0}^{o-1} \left[\binom{o-1}{m} \frac{(-1)^m}{n+ig-(o-1-m)}\right]^2 \\
&+ 2 \cdot \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} \binom{o-1}{m} \frac{(-1)^m}{n+ig-(o-1-m)} \binom{o-1}{j} \frac{(-1)^j}{n+ig-(o-1-j)} \\
=&\sum_{m=0}^{o-1} \binom{o-1}{m}^2 \left(\frac{1}{n-(o-1-m)+ig}\right)^2 \qquad ((-1)^m)^2 = 1 \text{ for all } m \in \mathbb{N}^+ \\
&+ 2 \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} \binom{o-1}{m} \binom{o-1}{j} \frac{(-1)^{m+j}}{(n-(o-1-m)+ig)(n-(o-1-j)+ig)}.
\end{aligned}$$

Substituting this back into the right side of Equation (16),

$$\begin{aligned}
&\sum_{i=0}^{k-1} o^2 \left(\sum_{m=0}^{o-1} \binom{o-1}{m} \frac{(-1)^m}{n+ig-(o-1-m)}\right)^2 \\
=&\sum_{i=0}^{k-1} o^2 \sum_{m=0}^{o-1} \binom{o-1}{m}^2 \left(\frac{1}{n-(o-1-m)+ig}\right)^2 \\
&+ \sum_{i=0}^{k-1} 2o^2 \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} \binom{o-1}{m} \binom{o-1}{j} \frac{(-1)^{m+j}}{(n-(o-1-m)+ig)(n-(o-1-j)+ig)} \\
=&o^2 \sum_{m=0}^{o-1} \binom{o-1}{m}^2 \cdot \sum_{i=0}^{k-1} \left(\frac{1}{n-(o-1-m)+ig}\right)^2 \\
&+ 2o^2 \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} (-1)^{m+j} \binom{o-1}{m} \binom{o-1}{j} \sum_{i=0}^{k-1} \frac{1}{(n-(o-1-m)+ig)(n-(o-1-j)+ig)}.
\end{aligned}$$

[11] For example, $(a+b+c)^2 = (a^2+ab+ac)+(ab+b^2+bc)+(ac+bc+c^2) = (a^2+b^2+c^2)+2(ab+ac+bc)$.

The first summation can be written

$$
\begin{aligned}
&o^2 \sum_{m=0}^{o-1} \binom{o-1}{m}^2 \cdot \sum_{i=0}^{k-1} \left(\frac{1}{n-(o-1-m)+ig} \right)^2 \\
=&\frac{o^2}{g^2} \sum_{m=0}^{o-1} \binom{o-1}{m}^2 \cdot \sum_{i=0}^{k-1} \left(\frac{g}{n-(o-1-m)+ig} \right)^2 \\
=&\frac{o^2}{g^2} \sum_{m=0}^{o-1} \binom{o-1}{m}^2 \left[\psi_1 \left(\frac{n-(o-1-m)}{g} \right) - \psi_1 \left(k + \frac{n-(o-1-m)}{g} \right) \right], \qquad (18)
\end{aligned}
$$

applying Lemma 4.5.6 with $A = n-(o-1-m)$ for the last equality. Applying Lemma 4.5.7 with $A = n-(o-1-m)$ and $B = n-(o-1-j)$, the other terms can be written

$$
\begin{aligned}
&2o^2 \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} (-1)^{m+j} \binom{o-1}{m} \binom{o-1}{j} \sum_{i=0}^{k-1} \frac{1}{(n-(o-1-m)+ig)(n-(o-1-j)+ig)} \\
=&2o^2 \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} (-1)^{m+j} \binom{o-1}{m} \binom{o-1}{j} \frac{1}{g[n-(o-1-m)-(n-(o-1-j))]} \\
&\qquad \cdot \left[\psi_0 \left(\frac{n-(o-1-m)}{g} \right) - \psi_0 \left(k + \frac{n-(o-1-m)}{g} \right) \right. \\
&\qquad \left. - \psi_0 \left(\frac{n-(o-1-j)}{g} \right) + \psi_0 \left(k + \frac{n-(o-1-j)}{g} \right) \right] \\
=&\frac{2o^2}{g} \sum_{m=0}^{o-1} \sum_{j=m+1}^{o-1} \frac{(-1)^{m+j}}{m-j} \binom{o-1}{m} \binom{o-1}{j} \\
&\qquad \cdot \left[\psi_0 \left(\frac{n-(o-1-m)}{g} \right) - \psi_0 \left(k + \frac{n-(o-1-m)}{g} \right) \right. \\
&\qquad \left. - \psi_0 \left(\frac{n-(o-1-j)}{g} \right) + \psi_0 \left(k + \frac{n-(o-1-j)}{g} \right) \right]. \qquad (19)
\end{aligned}
$$

The lemma follows by adding (18) and (19). □

4.5.4.3. *Approximating mean and variance of hypoexponentials.* The identities of Lemmas 4.5.8 and 4.5.9 give a way to compute the mean and variance of our hypoexponential distribution, defined by a sum with k terms, in time $o(k)$. Nevertheless, since empirical testing has shown that these identities require computing the digamma and trigamma functions at higher floating-point precision than standard 64-bit double precision, there are significant constant factors associated with this approach. In practice we actually approximate the mean and variance in the following much faster

way. Recall the mean of a hypoexponential defined by rates $\binom{n}{o}, \binom{n+g}{o}, \binom{n+2g}{o}, \ldots, \binom{n+(k-1)g}{o}$ is $\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}}$.

Define the *relative error* between numbers a and b to be $\frac{|a-b|}{\min(|a|,|b|)}$. Consider the first term $t_1 = \frac{1}{\binom{n}{o}}$ and last term $t_k = \frac{1}{\binom{n+(k-1)g}{o}}$ of the sum. We compute the relative error between t_1 and t_k, and if it is is less than 0.1,[12] we approximate the sum by using geometric mean of these terms:

$$\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}} \approx k \cdot \sqrt{\frac{1}{\binom{n}{o}} \cdot \frac{1}{\binom{n+(k-1)g}{o}}}.$$

The following lemma justifies this approximation, establishing a formal relationship between the relative error of the first and last terms t_1, t_k and the relative error between the actual sum and the approximation given using the geometric mean described above.

Lemma 4.5.10. *Let $\delta > 0$. If the relative error between $t_1 = 1/\binom{n}{o}$ and $t_k = 1/\binom{n+(k-1)g}{o}$ is δ, then the relative error between $\sum_{i=0}^{k-1} \frac{1}{\binom{n+ig}{o}}$ and $k\sqrt{t_1 t_k}$ is at most $\frac{\delta}{2} + \frac{\delta^2}{8}$.*

Proof. Let the terms be $t_i = 1/\binom{n+(i-1)g}{o}$ for $i \in \{1, \ldots, k\}$, so the sum is $\sum_{i=1}^{k} t_i$. Note that $t_1 > t_i > t_k$ for all $1 < i < k$; in the remainder of the proof, we use only this fact.

The relative error between t_1 and t_k is then $\delta = \frac{t_1 - t_k}{t_k} = \frac{t_1}{t_k} - 1$, implying $t_1 = t_k(1+\delta)$. Let $A = \frac{1}{k}\sum_{i=1}^{k} t_i$ be the arithmetic mean of the terms $t_1, \ldots, t_k$ (note kA is exactly their sum), and let $G = \left(\prod_{i=1}^{k} t_i\right)^{1/k}$ be the geometric mean of all k terms. Let $G_{1,k} = \sqrt{t_1 t_k}$, the geometric mean of the first and last term only. We first bound $|A - G|$, then $|G - kG_{1,k}|$.

From [17], with each $p_i = 1/k$ (or $1/n$ as stated in [17]; n there is k in this proof) gives

$$A - G \leq \frac{1}{2t_k} \sum_{i=1}^{k} \frac{1}{k} \left(t_i - \sum_{j=1}^{k} \frac{1}{k} t_j \right)^2.$$

Note that the outer sum on the right side is the sample variance $\mathrm{Var}\,[t_1, t_2, \ldots, t_k]$ of the terms, which we denote as $\mathrm{Var}\,[t]$, so we can write

$$A - G \leq \frac{\mathrm{Var}\,[t]}{2t_k}. \tag{20}$$

[12] Note that the larger is n, since $k \approx \sqrt{n}$, we expect the relative error between t_1 and t_k to be quite small, since it converges to 0 as $n \to \infty$. For example, if $n = 10^6, k = \sqrt{n} = 10^3, o = 2, g = 1$, then the relative error between t_1 and t_k is ≈ 0.002.

In the worst case, subject to only this constraint, the variance of the terms is maximized when half the terms are t_k and the other half are t_1. In that case, writing $\mu_{1,k} = (t_1 + t_k)/2$ for the (arithmetic) mean of those two terms, the variance would be

$$\frac{1}{k}\left[\frac{k}{2}(t_i - \mu_{1,k})^2 + \frac{k}{2}(t_k - \mu_{1,k})^2\right] = \frac{1}{4}(t_1 - t_k)^2,$$

which follows by substituting $\mu_{1,k} = (t_1 + t_k)/2$ and algebraic simplification. Thus $\mathrm{Var}\,[t] \leq \frac{1}{4}(t_1 - t_k)^2$. Since $t_1 = t_k(1+\delta)$, we have $t_1 - t_k = t_k(1+\delta) - t_k = t_k\delta$, Thus

$$\mathrm{Var}\,[t] \leq \frac{1}{4}(t_1 - t_k)^2 = \frac{t_k^2\delta^2}{4}.$$

Substituting in (20) gives $A - G \leq \frac{t_k\delta^2}{8}$.

Now we bound the geometric mean G of all terms using the geometric mean $G_{1,k} = \sqrt{t_1 t_k}$ of the first and last terms. Recall $t_1 = t_k(1+\delta)$, so

$$G_{1,k} = \sqrt{t_1 t_k} = \sqrt{t_k^2(1+\delta)} = t_k\sqrt{1+\delta}.$$

Note that $t_k \leq G \leq t_1 = t_k(1+\delta)$. If $G = t_k$, then

$$|G - G_{1,k}| = G_{1,k} - G = t_k\sqrt{1+\delta} - t_k = t_k(\sqrt{1+\delta} - 1)$$

and if $G = t_k(1+\delta)$, then

$$|G - G_{1,k}| = G - G_{1,k} = t_k\sqrt{1+\delta} - t_k = t_k(1+\delta) - t_k\sqrt{1+\delta} = \frac{t_k(\sqrt{1+\delta} - 1)}{\sqrt{1+\delta}}.$$

Since $\sqrt{1+\delta} > 1$ for all $\delta > 0$, this means the first case is larger, so in the worst case,

$$|G - G_{1,k}| \leq t_k(\sqrt{1+\delta} - 1)$$

Note that for all $\delta > 0$, we have $\sqrt{1+\delta} < 1 + \frac{\delta}{2}$. Combining the above bounds on $A - G$ and $|G - G_{1,k}|$ to bound $|A - G_{1,k}|$ by the triangle inequality:

$$|A - G_{1,k}| \leq |A - G| + |G - G_{1,k}| \leq \frac{t_k\delta^2}{8} + t_k\left(\sqrt{1+\delta} - 1\right) = \frac{t_k\delta^2}{8} + t_k\left(1 + \frac{\delta}{2} - 1\right) = t_k\left(\frac{\delta}{2} + \frac{\delta^2}{8}\right).$$

Recall the sum $S = \sum_{i=1}^{k} t_i = kA$. This implies our approximation by $kG_{1,k}$ has absolute error

$$|S - kG_{1,k}| \leq kt_k \left(\frac{\delta}{2} + \frac{\delta^2}{8}\right).$$

Recall $\sqrt{1+\delta} > 1$ for all $\delta > 0$. So the absolute error above implies relative error

$$\frac{|S - kG_{1,k}|}{kG_{1,k}} \leq \frac{kt_k\left(\frac{\delta}{2} + \frac{\delta^2}{8}\right)}{k\sqrt{t_1 t_k}} = \frac{t_k\left(\frac{\delta}{2} + \frac{\delta^2}{8}\right)}{\sqrt{t_k^2(1+\delta)}} = \frac{\frac{\delta}{2} + \frac{\delta^2}{8}}{\sqrt{1+\delta}} < \frac{\delta}{2} + \frac{\delta^2}{8}. \qquad \square$$

We in fact conjecture that the relative error of the approximation of the sum by $kG_{1,k}$ is $O(\delta^2)$ (better than merely $O(\delta)$ as in Lemma 4.5.10), which appears to be the case empirically. For example, when the relative error of t_1 and t_k is 0.1, in practice the relative error of the approximation and the true mean appears to be < 0.01. However, we have not been able to prove this. It would likely require using more information about the distribution of the terms t_i; under our worst case assumption in the proof that G could be as small as t_k (i.e, assuming all terms in the sum are t_k) or as large as t_1 (i.e, assuming all terms in the sum are t_1), the bound proven seems asymptotically tight without additional constraints on the terms.

We similarly approximate the variance $\sum_{i=1}^{k} t_i^2$ via $k\sqrt{t_1^2 t_k^2} = kt_1 t_k$, though we omit a detailed analysis. A similar proof to that of Lemma 4.5.10 shows that it is also bounded by the relative error between the first and last terms. Since these terms are squared and significantly less than 1, the relative error between t_1^2 and t_k^2 is even smaller than in the case of approximating the mean, making this an even tighter approximation of the variance than that of the mean in Lemma 4.5.10. In practice, these approximations appear to result in sampling CRN trajectories indistinguishable from the Gillespie algorithm.

4.6. Open Questions

There are two pressing theoretical open questions relating to asymptotic efficiency. The first pertains to slowdown described in Definition 4.2.8 from probabilistic reactions on certain configurations (e.g., $2L \to L + F$ as $\#L \to 0$). It is possible that any exact algorithm that chooses reactions by selecting individual reactants will suffer from this slowdown, as it is necessary to ensure correct reaction probabilities in configurations where a high-propensity reaction has one or more low-count reactants. There may be some way to utilize the core idea of batching, but choose which reactants

comprise a batch in a more clever way. It may also be possible to choose reactions in some other way than either choosing individual reactions (as in Gillespie) or choosing individual reactants (as in batching).

The second theoretical question pertains to the asymptotic cost of adaptive rejection sampling. Ideally, in all situations, our algorithm would be able to simulate $\Theta(\sqrt{n})$ reactions in $O(\log n)$ time; however, when sampling from the hypoexponential exactly using adaptive rejection sampling, this only yields optimal efficiency when $\ell \geq n^{5/4}$. Future work may show how to avoid this by exactly sampling timestamps more efficiently.

Another question is how broadly applicable this algorithm is beyond CRNs. It may be possible to adapt the algorithm to other stochastic processes that select elements from some set at each iteration to undergo a "transition", but are not modeled exactly by the Gillespie algorithm, for example surface CRNs [19] or tile displacement systems [52, 67], which are both chemical models that account for geometrical arrangement of some chemical species that interact with each other.

Practically, it may also be useful to develop approximate simulation algorithms that derive their logic from our exact simulation algorithm. For example, by skipping the step of sampling collision-free run length and instead simply choosing the run length to be its expected value, one can avoid the numerical precision issues that our implementation has. It seems feasible that such practical algorithms may still be amenable to theoretical analysis of bound approximation accuracy as in e.g. [57].

CHAPTER 5

Chemical Reaction Network Simulator Implementation

This chapter is joint work with David Doty, built from the ppsim package [24] written by David Doty and Eric Severson. [46]. In order to verify the theory in the previous chapter, we implemented it. We will soon make our implementation available publicly as a python package. Our implementation is written primarily in rust for efficiency. Here we provide simulation data obtained by running our algorithm on some CRNs, in order to show that the algorithm is correct (i.e., matches the Gillespie algorithm in distribution) and efficient.

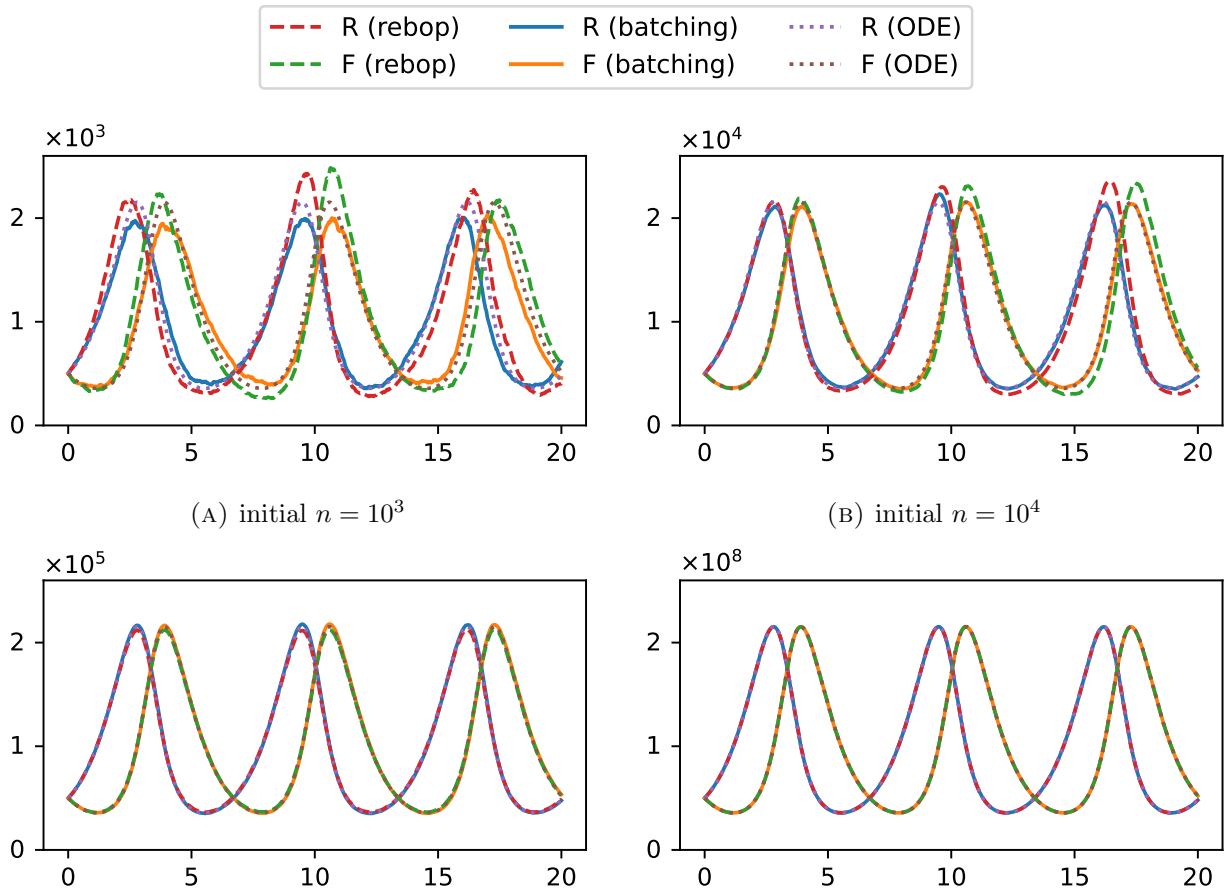
5.1. Simulation data

5.1.1. Empirical sampling of data for some CRNs. This section shows data for a few CRNs that are best suited for using our batching algorithm; namely they have positive generativity (hence cannot be simulated by the original batching algorithm) but are order 2, hence have the fewest problems with passive reactions as described in Algorithm 2. More formally, CRNs with smaller values of $S_{\mathcal{C},v}(\mathbf{c})$ as defined in Definition 4.2.8.

As in Section 5.1.3, when we compare our batching algorithm to Gillespie, we use rebop [3] as the fastest Gillespie algorithm implementation that we could find.[1] We note that rebop does not directly support the concept of volume v; it implicitly assumes $v = 1$. Recall in Section 4.2.1 that each reaction with o total reactants has a term $1/v^{o-1}$ in its rate. So for proper comparison, we manually adjust the rebop rate constants in this way (i.e., divide order-o reaction rate constants by v^{o-1}) so that rebop's reactions have the same total rate as reactions in our batching algorithm.

We note that deviations in simulated trajectories do not imply that the batching algorithm is sampling from the wrong distribution. Rather, in all cases this is simply stochasticity of the Gillespie model itself; see Figure 5.2b for further empirical justification of the claim that our algorithm samples from the Gillespie distribution.

[1] Rebop is *far* faster than most Gillespie implementations: https://github.com/Armavica/rebop#performance



(A) initial $n = 10^3$

(B) initial $n = 10^4$

(C) initial $n = 10^5$

(D) initial $n = 10^8$

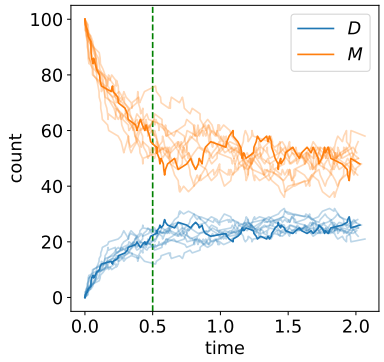
FIGURE 5.1. Plots of counts vs. time for the Lotka-Volterra oscillator CRN, for initial population size $n \in \{10^3, 10^4, 10^5, 10^8\}$ with half predator, half prey, using both rebop (Gillespie algorithm) and our batching algorithm implementation. Stochastic effects make the plots behave differently for small n, as well as differently from a deterministic ODE approximation, also plotted. As n increases, stochastic noise decreases, and both stochastic simulators generate trajectories approaching the deterministic model. At $n = 10^8$ all three plots are nearly indistinguishable.

Figure 5.1 shows simulations of the Lotka-Volterra chemical oscillator [41,61], a.k.a., predator-prey oscillator: the reactions

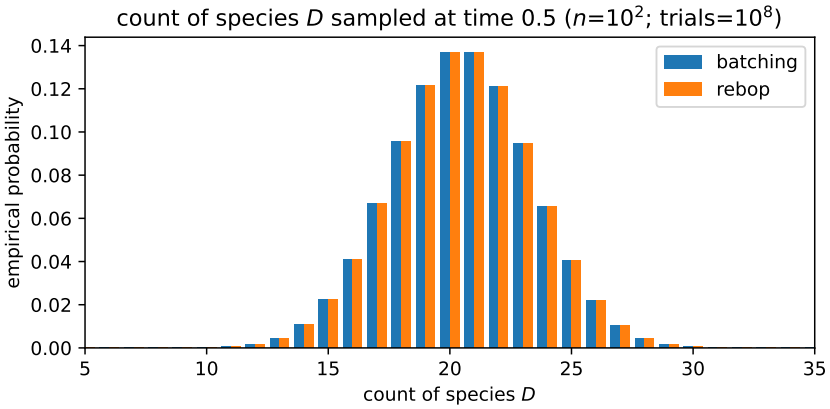
$$R \xrightarrow{1} 2R$$

$$F \xrightarrow{1} \emptyset$$

$$F + R \xrightarrow{1} 2F$$



(A) Plot of species counts vs. time in ten example runs.



(B) Comparing distribution of batching algorithm to Gillespie (rebop).

FIGURE 5.2. Figure 5.2a shows a plot of ten example runs of the reversible dimerization reaction $2M \underset{1}{\overset{1}{\rightleftharpoons}} D$, starting with $\#M = 100$ and $\#D = 0$, showing counts vs time for each run. The CRN approaches an equilibrium with expected 25 copies of D and expected 50 copies of M, typically reaching there after about 1 unit of time. (Though of course the counts bounce around even at equilibrium.) Figure 5.2b shows a plot of empirical distributions from rebop and the batching algorithm, of the count of D at time 0.5, just prior to (likely) convergence, where $\mathrm{E}[\#D] \approx 20.5$. Here, "empirical probability" means the total number of runs in which the given count was the count of D at time 0.5, divided by the total number of trials.

starting with equal R (rabbits/prey) and F (foxes/predators). Although devised originally by Lotka to study chemical reaction networks with autocatalytic reactions [41], this model was independently devised by Volterra [61] to model actual animal populations that were observed empirically to oscillate over the timescale of years. The intuition is that rabbits always have plenty of plants to eat, so constantly reproduce ($R \to 2R$), foxes die if they are hungry ($F \to \emptyset$), but foxes reproduce if they eat a rabbit ($F + R \to 2F$).

5.1.2. Empirically the batching algorithm samples from the Gillespie distribution.

Figure 5.2 demonstrates empirically that the batching algorithm samples from the same distribution as the Gillespie algorithm. We run both our batching algorithm and rebop on the CRN with the reversible dimerization reaction $2M \underset{1}{\overset{1}{\rightleftharpoons}} D$, where two monomers M can join to form an unstable dimer D, which can in turn split back into monomers. For the sake of collecting many samples, we choose a small initial population size $n = 100$: start with $\#M = 100$. Run until time 0.5, and measure the count of D, for many trials. We plot the empirical distribution (number of times D had the given count, divided by the number of trials) for both our batching algorithm and rebop.

We chose the reversible dimerization CRN $2M \rightleftharpoons D$ because, unlike the Lotka-Volterra CRN used for other examples, the dimerization CRN cannot "go extinct." The Lotka-Volterra CRN has the unfortunate property that, if rabbits die out, then so do foxes eventually, leading to a so-called "terminal" configuration in which no reactions are applicable. Many CRN simulators, such as rebop, simply hang if asked to simulate to a certain time, if they go terminal before that time is reached.

Conceptually, this is not difficult to deal with, but in practice it is easier simply to simulate a CRN with no reachable terminal configurations. Thus we use the CRN $2M \rightleftharpoons D$, which cannot go terminal, nor can its molecular count increase without bound. Nevertheless, as with Lotka-Volterra, it represents an excellent test case for our new algorithm, since it fundamentally requires a reaction with positive generativity, the key challenge in adapting the batching algorithm of [7] to more general CRNs.

5.1.3. Batching algorithm has quadratic speedup over Gillespie algorithm. Figure 5.3 shows runtime scaling of our batching algorithm vs. the Gillespie algorithm, as implemented by the rebop package [3], which is the fastest Gillespie implementation that we have found. We tested against both the rebop Python package [3], and the rebop Rust crate [4] where one can implement the CRN in a pure Rust program.

The "kink" in the batching f128 runtime at $n = 10^{11}$ of Figure 5.3 corresponds to an implementation issue: Our way of sampling the length of a collision-free run involves several floating-point operations, mainly to repeatedly compute the log-gamma function $\ln \Gamma(x)$. On sufficiently large population sizes, we found empirically that standard 64-bit double precision floats lacked the precision to compute these numbers in rare extreme cases. In particular, our method of inversion sampling samples a float uniformly in the unit interval $u \in [0, 1)$. We compute $\ln u$. In computing the CDF of the distribution in order to do inversion sampling, this number $\ln u$ is compared to numbers computed as the difference of log-gamma of much larger numbers, scaling with population size n. When the log-gamma of those large numbers (recall $\ln \Gamma(n) \approx n \ln n$) is sufficiently large (in this case, empirically around $n \geq 10^{11}$), the imprecision causes the inversion sampling to be incorrect, because the differences between those large numbers are insufficiently precise to compare meaningfully to $\ln u$.

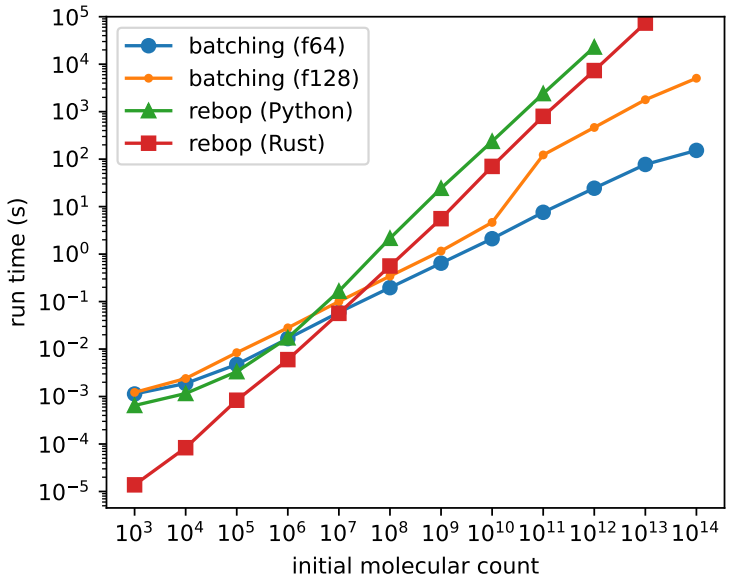


FIGURE 5.3. Runtime scaling of our batching algorithm, using both standard double precision floating-point arithmetic (f64), and quadruple precision (f128), vs. the Gillespie algorithm, as implemented by the rebop Python package [3], and the pure Rust rebop crate [4]. With the Lotka-Volterra reactions, each is simulated on the given initial population size n (with half predators and half prey) until time 1.0, which corresponds to $\Theta(n)$ total reactions. As expected, rebop shows asymptotic scaling of $\Theta(n)$ time (slope 1 on a log-log plot) compared to scaling of $\Theta(\sqrt{n})$ for batching (slope 1/2). The "kink" in the batching f128 run time at $n = 10^{11}$ is explained in the main text and is due to extra floating-point precision used in some calculations only in larger population sizes.

To correct for this, we use Rust's experimental f128 type that has so-called "quadruple" precision, double that of a standard double (called f64 in Rust). However, because hardware and OS support for f128 is incomplete, we had to implement our own slower software implementation of the natural log function and the log-gamma function on f128 values. This is the source of the observed slowdown when $n \geq 10^{11}$. In practice, even if we allow these errors through and just use the faster f64 implementation (also shown in Figure 5.3), there does not appear to be any systematic bias that leads the sampled CRN trajectory to appear sufficiently different from the correct Gillespie distribution. The runtime using that optimization is shown in Figure 5.3. However, we kept the slow f128 implementation as the standard to ensure correctness. We believe that this can be optimized in practice, and that the f64 data shows something more representative of what we believe is possible in practice, but we have yet to thoroughly investigate this.

It is also worth noting that even for $n < 10^{11}$, the running times for batching are slightly larger for what is labeled f128 in Figure 5.3 than f64. This is because some simple calculations, for example adding and subtracting numbers, are done with f128 precision unconditionally (where it would take at least twice as long just due to having twice as many bits to process, since there is currently scant hardware support for single-instruction f128 operations), whereas other more complex operations (because of the nearly 10x slowdown observed in the case $n \geq 10^{11}$) only use f128 precision when certain values are sufficiently large that the extra precision is warranted.

More precisely, all of this happens when we sample the length of a collision-free run. The algorithm described in the proof of Lemma 4.4.6 involves summing terms that are the output of the log-gamma function. These outputs are always represented as f128 values, which take roughly twice as long to add or subtract as f64 values, accounting for the unconditional slowdown in all population sizes in f128 compared to f64. However, for sufficiently small values as input to log-gamma, for efficiency we compute log-gamma using f64 values, even though we then store that output in an f128 to add the terms, and in this case $n \leq 10^{10}$ will imply that case. The additional slowdown for $n \geq 10^{11}$ is because the computation of log-gamma itself switches to using f128 values internally, and this computation is sufficiently complex that the constant factor increase here is more noticeable; in particular, computing the natural log function is a significant factor in computing log-gamma, but natural log for f128 is not supported yet in Rust, so we had to write our own software implementation of natural log as well.

Regarding the performance of the rebop Python package versus the rebop pure Rust crate: the rebop documentation claims

> *Performance and ergonomics are taken very seriously. For this reason, two independent APIs are provided to describe and simulate reaction networks:*
>
> - *a macro-based DSL implemented by* [`define_system`], *usually the most efficient, but that requires to compile a rust program;*
> - *a function-based API implemented by the module* [`gillespie`], *also available through Python bindings. This one does not require a rust compilation and allows the system to be defined at run time. It is typically 2 or 3 times slower than the macro DSL, but still faster than all other software tried.*

The superior performance of the pure Rust crate ("macro-based DSL") over the Python API is evident in the difference between the data labeled "rebop (Python)" and "rebop (Rust)" in Fig. 5.3. Our package, like the rebop Python package, uses a Rust backend with a Python front-end using PyO3 to call Rust from Python. Since we intend our package as a Python package, which will be much easier to use for most users, it seems that the fair comparison is to the rebop Python API, rather than the faster pure Rust crate. Yet, whether comparing to the rebop Python API or Rust API, in either case the linear scaling (time $\Theta(n)$ to simulate n reactions) shown in Figure 5.3 still holds, but the pure Rust implementation of rebop stays superior to our batching algorithm for slightly large population sizes than the rebop Python package. Nevertheless the batching algorithm is clearly superior to both for $n \geq 10^8$.

We intend eventually to implement a pure Rust crate for our batching algorithm. We may see a performance gain as well, but it is not clear that we can do the same optimizations done by rebop, which somehow optimizes to get a constant factor speedup based on knowing the reactions at compile time. In the case of the batching algorithm, there's no obvious optimization that can be done based on knowing the reactions at compile time. The constant-factor overhead of rebop's Python API is definitely not merely the overhead of calling from Python, because the times for large population sizes were measured with a single call to rebop's `Gillespie.run` method (called from Python, but implemented as a Rust method), which has only a small additive constant overhead to call once, yet the performance of the pure Rust implementation remains a constant factor faster no matter how large n.

5.1.3.1. *Multibatching.* The batching algorithm of [7] has an optimized version called "multi-batching", in which multiple collisions are simulated in a single batch. This is useful when the time t_c to sample the collision-free run length and simulate a collision is less than the time t_b required to process a batch; if $t_b \approx t_c \cdot m$ for $m \in \mathbb{N}^+$, then the ideal tradeoff would be to simulate m collisions for each batch. This is the main reason it is useful to have a definition of the collision-free run length distribution $\mathbf{coll}(n, r, o, g)$ that allows $r > 0$, even though here we focused on the case $r = 0$, since $r > 0$ is how we model the question "*given that r molecules have already interacted from previous collisions and the batch length up to this point, how many* additional *molecules can be picked before another collision?*", and then that number is added to the growing batch length.

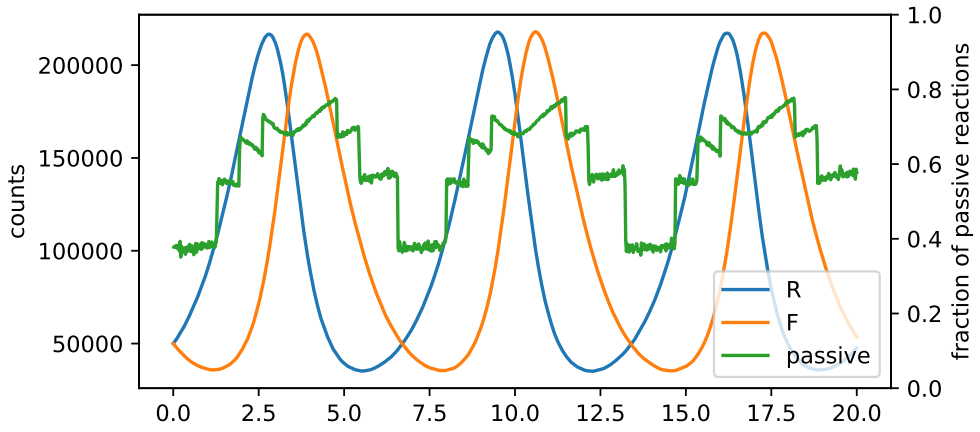


FIGURE 5.4. Plot of counts vs. time for the Lotka-Volterra CRN with initial molecular count $n = 10^5$, until time 20; with the fraction of passive reactions also shown (note separate y-axis on right).

In other words, instead of a single step of the algorithm being to process a batch of length $\ell \sim \mathbf{coll}(n, 0, o, g)$, then simulate a single collision, we instead imagine the following process. We sample a collision-free run length $\ell_1 \sim \mathbf{coll}(n, 0, o, g)$, then a second collision-free run length $\ell_2 \sim \mathbf{coll}(n, (\ell_1 + 1) \cdot (o + g), o, g)$ (to model that the first $\ell_1 + 1$ reactions (ℓ_1 in the first part of the batch, plus 1 for the collision) turn $(\ell_1 + 1) \cdot (o + g)$ total molecules "red". Then we sample a third collision-free run length $\ell_3 \sim \mathbf{coll}(n, (\ell_1 + \ell_2 + 2) \cdot (o + g), o, g)$, etc., up to ℓ_m. We then simulate the m interactions involving the m collisions (see [7, Section 4] for details), and a single batch of length $\sum_{i=1}^{m} \ell_i$.

Although it is possible to implement multibatching for our generalized batching algorithm, we have not explored the idea in depth. Empirically our implementation spends the bulk of its time sampling the collision-free run length, due mostly to the extra complexity of our definition of a collision, generalized to allow positive generativity. If this could be optimized to take far less time, then implementing a similar multibatching approach in our algorithm would become beneficial.

5.1.3.2. *Passive reactions.* Recall that a potential source of slowdown for the batching algorithm is simulating many passive reactions (see Algorithm 2), which do not correspond to reactions executed in the original CRN. The greater the fraction of passive reactions in a batch, the less progress the batching algorithm makes toward simulating the original CRN.

To measure empirically how many reactions are passive, Figure 5.4 shows Lotka-Volterra CRN with initial population size $n = 10^5$, with counts plotted alongside the fraction of passive reactions

sampled at each time point. The discrete jumps in this plot correspond to the population size n changing by a significant enough fraction (0.7 in our implementation) that we reset the count of species K to be the new value of n. Because $\#K$ influences correction factors in rate constants (see Definition 4.2.7 and Algorithm 1), changing $\#K$ requires recomputing rate constants. We avoid changing K on every step because it is computationally prohibitive to recompute the rate constants that frequently.

Currently we set $\#K = n$ (i.e., equal to the number of molecules in the original CRN) when we reset $\#K$, since this gives the asymptotic behavior that we want of maintaining a $\Omega(1)$ constant fraction of sampled reactions are non-passive. However, it remains to optimize; the optimal choice of $\#K$ is $\Theta(n)$, but choosing the actual value more carefully could potentially reduce the number of null reactions. Given the plot in Figure 5.4, where the fraction of passive reactions hovers under 0.8, this means we ideally hope for a speedup of no greater than 5x for this particular example.

Bibliography

[1] mpmath Python package. URL: https://mpmath.org/.

[2] nuad Python package. URL: https://github.com/UC-Davis-molecular-computing/nuad.

[3] rebop Python package. https://pypi.org/project/rebop/ https://github.com/Armavica/rebop.

[4] rebop Rust crate. https://docs.rs/rebop/latest/rebop/ https://github.com/Armavica/rebop.

[5] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55. Courier Corporation, 1965.

[6] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, March 2006.

[7] Petra Berenbrink, David Hammer, Dominik Kaaser, Ulrich Meyer, Manuel Penschuck, and Hung Tran. Simulating Population Protocols in Sub-Constant Time per Interaction. In *ESA 2020: 28th Annual European Symposium on Algorithms*, volume 173, pages 16:1–16:22, 2020. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ESA.2020.16, doi:10.4230/LIPIcs.ESA.2020.16.

[8] Jose M Bernardo. Algorithm as 103: Psi (digamma) function. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 25(3):315–317, 1976.

[9] C. E. Blair and R. G. Jeroslow. The value function of an integer program. *Mathematical Programming*, 23(1):237–273, Dec 1982. doi:10.1007/BF01583794.

[10] Kimiko O Bowman and LR Shenton. Estimation: Method of moments. *Encyclopedia of Statistical Sciences*, 3, 2004.

[11] Keenan Breik, Cameron Chalk, David Doty, David Haley, and David Soloveichik. Programming substrate-independent kinetic barriers with thermodynamic binding networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 18(1):283–295, 2021. doi:10.1109/TCBB.2019.2959310.

[12] Keenan Breik, Chris Thachuk, Marijn Heule, and David Soloveichik. Computing properties of stable configurations of thermodynamic binding networks. *Theoretical Computer Science*, 785:17–29, 2019.

[13] Xiaodong Cai and Ji Wen. Efficient exact and k-skip methods for stochastic simulation of coupled chemical reactions. *The Journal of Chemical Physics*, 131(6), 2009.

[14] Yang Cao, Daniel T Gillespie, and Linda R Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics*, 124(4), 2006.

[15] Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. Stochastic analysis of chemical reaction networks using linear noise approximation. *Biosystems*, 149:26–33, 2016. Selected papers from the Computational Methods in Systems Biology 2015 conference. URL: https://www.sciencedirect.com/science/article/pii/S0303264716302039, doi:10.1016/j.biosystems.2016.09.004.

[16] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Forward and backward bisimulations for chemical reaction networks. In *CONCUR 2015*, 2015.

[17] DI Cartwright and MJ Field. A refinement of the arithmetic mean-geometric mean inequality. *Proceedings of the American Mathematical Society*, pages 36–38, 1978.

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

[19] Samuel Clamons, Lulu Qian, and Erik Winfree. Programming and simulating chemical reaction networks on a surface. *Journal of the Royal Society Interface*, 17(166):20190790, 2020.

[20] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[21] Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

[22] David Doty, Benjamin L Lee, and Tristan Stérin. scadnano: A browser-based, scriptable tool for designing dna nanostructures, 2020. URL: https://arxiv.org/abs/2005.11841, arXiv:2005.11841.

[23] David Doty, Trent A. Rogers, David Soloveichik, Chris Thachuk, and Damien Woods. Thermodynamic binding networks. In Robert Brijder and Lulu Qian, editors, *DNA Computing and Molecular Programming*, pages 249–266, Cham, 2017. Springer International Publishing.

[24] David Doty and Eric Severson. ppsim: A software package for efficiently simulating and visualizing population protocols. In *CMSB 2021: Proceedings of the 19th International Conference on Computational Methods in Systems Biology*, pages 245–253, 2021. URL: https://arxiv.org/abs/2105.04702.

[25] Giulia Fanti, Nina Holden, Yuval Peres, and Gireeja Ranade. Communication cost of consensus for nodes with limited memory. *Proceedings of the National Academy of Sciences*, 117(11):5624–5630, 2020.

[26] Martin Feinberg. *Foundations of Chemical Reaction Network Theory*, volume 202. Springer, 2019.

[27] Mark E Fornace, Jining Huang, Cody T Newman, Nicholas J Porubsky, Marshall B Pierce, and Niles A Pierce. Nupack: Analysis and design of nucleic acid structures, devices, and systems. 2022.

[28] Michael A Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.

[29] Walter R Gilks and Pascal Wild. Adaptive rejection sampling for Gibbs sampling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 41(2):337–348, 1992.

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

[31] Daniel T Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001.

[32] Daniel T Gillespie. Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, 58(1):35–55, 2007.

[33] Cato M. Guldberg and Peter Waage. Studies concerning affinity. *Forhandlinger: Videnskabs-Selskabet i Christinia. Norwegian Academy of Science and Letters*, 35, 1864. English translation in [**64**].

[34] David Haley and David Doty. Computing properties of thermodynamic binding networks: An integer programming approach. In Matthew R. Lakin and Petr Šulc, editors, *DNA 2021: Proceedings of the 27th International Meeting on DNA Computing and Molecular Programming*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14669, doi:10.4230/LIPIcs.DNA.27.2.

[35] Robert Johnson, Qing Dong, and Erik Winfree. Verifying chemical reaction network implementations: A bisimulation approach. *Theoretical Computer Science*, 765:3–46, 2019.

[36] Matthew Johnston. *Topics in chemical reaction network theory*. PhD thesis, University of Waterloo, 2012.

[37] Thomas G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *The Journal of Chemical Physics*, 57(7):2976–2978, 1972.

[38] James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Hugh D. Potter, and Matthew R. Riley. Population-Induced Phase Transitions and the Verification of Chemical Reaction Networks. In Cody Geary and Matthew J. Patitz, editors, *26th International Conference on DNA Computing and Molecular Programming (DNA 26)*, volume 174 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.DNA.2020.5, doi:10.4230/LIPIcs.DNA.2020.5.

[39] Randolph Lopez, Ruofan Wang, and Georg Seelig. A molecular multi-gene classifier for disease diagnostics. *Nature chemistry*, 10(7):746–754, 2018.

[40] Ronny Lorenz, Stephan H Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. Viennarna package 2.0. *Algorithms for molecular biology*, 6(1):26, 2011.

[41] Alfred J Lotka. Contribution to the theory of periodic reactions. *The Journal of Physical Chemistry*, 14(3):271–274, 1910.

[42] Daniel W Lozier. NIST digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence*, 38:105–119, 2003. URL: https://dlmf.nist.gov/5.15.

[43] Ernst W. Mayr and Jeremias Weihmann. A framework for classical petri net problems: Conservative petri nets as an application. In Gianfranco Ciardo and Ekkart Kindler, editors, *Application and Theory of Petri Nets and Concurrency*, pages 314–333, Cham, 2014. Springer International Publishing.

[44] Dionis Minev, Christopher M Wintersinger, Anastasia Ershova, and William M Shih. Robust nucleation control via crisscross polymerization of highly coordinated DNA slats. *Nature communications*, 12(1):1741, 2021.

[45] Eric Mjolsness, David Orendorff, Philippe Chatelain, and Petros Koumoutsakos. An exact accelerated stochastic simulation algorithm. *The Journal of Chemical Physics*, 130(14), 2009.

[46] Joshua Petrack and David Doty. Exactly simulating stochastic chemical reaction networks in sub-constant time per reaction, 2025. URL: https://arxiv.org/abs/2508.04079, arXiv:2508.04079.

[47] Joshua Petrack, David Soloveichik, and David Doty. Thermodynamically driven signal amplification. *arXiv preprint arXiv:2307.01550*, 2023.

[48] William H Press, BP Flannery, SA Teukolsky, and WT Vetterling. Section 17.1 Runge-Kutta method. *Numerical Recipes: The Art of Scientific Computing*, 2007.

[49] M. I. Qureshi and Mohd Shadab. Analytic computations of digamma function using some new identities, 2018. URL: https://arxiv.org/abs/1806.07948, arXiv:1806.07948.

[50] Muruhan Rathinam and Hana El Samad. Reversible-equivalent-monomolecular tau: A leaping method for "small number and stiff" stochastic chemical systems. *Journal of Computational Physics*, 224(2):897–923, 2007.

[51] Sheldon M Ross. *Introduction to Probability Models*. Academic press, 2014.

[52] Namita Sarraf, Kellen R Rodriguez, and Lulu Qian. Modular reconfiguration of DNA origami assemblies using tile displacement. *Science Robotics*, 8(77):eadf1511, 2023.

[53] Gerald Schochetman, Chin-Yih Ou, and Wanda K. Jones. Polymerase chain reaction. *The Journal of Infectious Diseases*, 158(6):1154–1157, 1988. URL: http://www.jstor.org/stable/30137034.

[54] Georges Schwachheim. Algorithm 349: polygamma functions with arbitrary precision [s14]. *Communications of the ACM*, 12(4):213–214, April 1969. doi:10.1145/362912.362928.

[55] Seung Woo Shin, Chris Thachuk, and Erik Winfree. Verifying chemical reaction network implementations: A pathway decomposition approach. *Theoretical Computer Science*, 765:67–96, 2019.

[56] Alexander Slepoy, Aidan P Thompson, and Steven J Plimpton. A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics*, 128(20):05B618, 2008.

[57] David Soloveichik. Robust stochastic chemical reaction networks and bounded tau-leaping. *Journal of Computational Biology*, 16(3):501–522, 2009.

[58] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010. URL: https://www.pnas.org/doi/abs/10.1073/pnas.0909380107, arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.0909380107, doi:10.1073/pnas.0909380107.

[59] John L. Spouge. Computation of the gamma, digamma, and trigamma functions. *SIAM Journal on Numerical Analysis*, 31(3):931–944, 1994. doi:10.1137/0731050.

[60] Ernst Stadlober. Ratio of uniforms as a convenient method for sampling from classical discrete distributions. In *Proceedings of the 21st conference on Winter simulation*, pages 484–489, 1989.

[61] Vito Volterra. *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. Società anonima tipografica "Leonardo da Vinci"', 1926.

[62] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.

[63] Petr Šulc, Flavio Romano, Thomas E. Ouldridge, Lorenzo Rovigatti, Jonathan P. K. Doye, and Ard A. Louis. Sequence-dependent thermodynamics of a coarse-grained dna model. *The Journal of Chemical Physics*, 137(13):135101, 2012. URL: http://link.aip.org/link/?JCP/137/135101/1, doi:10.1063/1.4754132.

[64] Peter Waage and Cato Maximilian Gulberg. Studies concerning affinity. *Journal of chemical education*, 63(12):1044, 1986. English translation; original paper is [**33**].

[65] RO Weber and SI Barry. Finite-time blow-up in reaction-diffusion equations. *Mathematical and computer modelling*, 18(10):163–168, 1993.

[66] Pascal Wild and WR Gilks. Algorithm as 287: Adaptive rejection sampling from log-concave density functions. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 42(4):701–709, 1993.

[67] Erik Winfree and Lulu Qian. Two-dimensional tile displacement can simulate cellular automata. *arXiv preprint arXiv:2301.01929*, 2023.

[68] Erhu Xiong, Dongbao Yao, Andrew D. Ellington, and Sanchita Bhadra. Minimizing leakage in stacked strand exchange amplification circuits. *ACS Synthetic Biology*, 10(6):1277–1283, 2021. PMID: 34006090. arXiv:https://doi.org/10.1021/acssynbio.0c00615, doi:10.1021/acssynbio.0c00615.