Computing properties of stable configurations of thermodynamic binding networks

Keenan Breik[1], Chris Thachuk[2], Marijn Heule[1], David Soloveichik[1]

[1] University of Texas at Austin
[2] California Institute of Technology

Abstract. The promise of chemical computation lies in controlling systems incompatible with traditional electronic micro-controllers, with applications in synthetic biology and nano-scale manufacturing. Computation is typically embedded in kinetics—the specific time evolution of a chemical system. However, if the desired output is not thermodynamically stable, basic physical chemistry dictates that thermodynamic forces will drive the system toward error throughout the computation. The thermodynamic binding network (TBN) model was introduced to formally study how the thermodynamic equilibrium can be made consistent with the desired computation, and it idealizes tradeoffs between configurational entropy and binding. Here we prove the computational hardness of natural questions about TBNs and develop a practical algorithm for verifying the correctness of constructions by translating the problem into propositional logic and solving the resulting formula. The TBN model together with automated verification tools will help inform strategies for error reduction in molecular computing, including the extensively studied models of strand displacement cascades and algorithmic tile assembly.

Keywords: chemical computation, hardness of approximation, reduction to SAT.

1 Introduction

Similar to digital electronics, advances in engineering of molecular computation have relied on a distinctive set of abstractions and models. The formalism of algorithmic tile assembly [8] enabled the self-assembly of complex nanostructures from simple parts, with a molecular computational process (e.g., simulating binary counting) directing component placement [2, 17]. Likewise, DNA strand displacement cascades (formalized as [18]) made it possible to rationally design molecular reaction pathways, and this model has been used to engineer a wide range of molecular devices, programmable structures, logic and neural circuits, and dynamical systems [24, 22, 4]. The ultimate applications of molecular computation are in contexts where traditional electronics cannot be used. These applications include reprogramming biological cell behaviors or controlling complex nanoscale assembly processes. It is also hoped that theories of molecular computing can

shed light on ill-understood design principles of natural biological regulatory and development pathways.

The widely studied models of chemical computing such as algorithmic tile assembly and strand displacement cascades are essentially kinetic as they describe a desired time evolution of an information processing chemical system. However, unlike electronic computation, chemical computation operates in a Brownian environment subject to powerful thermodynamic driving forces. If the desired output happens to be a meta-stable configuration, then thermodynamic driving forces will inexorably drive the system toward error. For example, in tile assembly, thermodynamically favored assemblies that are not the intended self-assembly program execution are likewise a major source of error [19, 2]. Likewise, *leak* in most strand displacement systems occurs because the thermodynamic equilibrium of a strand displacement cascade favors incorrect over the correct output, or does not discriminate between the two [23].

The thermodynamic binding network (TBN) model abstracts chemical systems at the thermodynamic equilibrium [9]. The model is simple and general due to its two main features: (1) abstracting away of "geometry", and (2) the simplification of thermodynamics to a tradeoff between the number of separate complexes (configurational entropy), and the number of bonds formed. These features of the model make it widely applicable, including for understanding the consistency of kinetics and thermodynamics for strand displacement cascades, algorithmic tile assembly, as well as other contexts. The simplicity of the model makes it amenable to rigorous proofs.

The most basic question is how can we distinguish (thermodynamically) *stable* configurations—that is configurations that are energetically favorable—from unstable ones. Predicting the thermodynamic equilibrium is often computationally difficult: for example, determining the lowest energy configurations of Ising models [5], predicting secondary structure of nucleic-acids with pseudoknots [15], and predicting protein folding [11] were shown to be NP-hard. These problems derive their hardness from geometrical constraints; in contrast the TBN model avoids geometry, and the computational complexity originates in the interplay between the opposing forces of increasing binding and increasing the number of separate complexes. Even without configurational entropy, there are interesting computational problems derived solely from geometry-free binding [14].

In the TBN model, the hard part of checking whether a given configuration is stable is determining whether the system can reconfigure to increase the number of polymers, thereby increasing configurational entropy, without reducing the total number of bonds. We prove that this problem is NP-complete in the worst case, and also that the problem of computing the number of polymers in a stable configuration is not in n^{δ}-APX for any $\delta < 1$ (that is, it is hard to approximate).

In DNA strand displacement cascades, output is usually represented by the release of a previously bound DNA strand. The question of whether releasing the output is thermodynamically favorable corresponds to the problem of deciding whether a given monomer is free in some stable configuration of the TBN model. We show that this problem is complete for $\mathrm{P}_{\|}^{\mathrm{NP}}$ (which is P with parallel access

to an NP oracle), making it one of very few known natural complete problems for this class.

Despite these worst-case negative results, we develop a software package accompanying this paper that can answer many questions in practice [1]. The package computes a non-trivial reduction to the boolean satisfiability problem (SAT) which is then passed to a SAT solver. An exponential speed up can be achieved in certain cases, compared with a naive solution based on enumerating all configurations, or even the subset of maximally bound (saturated) configurations. Our package assists in manipulating and understanding the behavior of TBNs.

Our ultimate goal is molecular computation in which the thermodynamic equilibrium is consistent with the desired computation, and that it can be reached by an efficient kinetic pathway. The TBN model permits differentiating thermodynamic and kinetic contributions to the computational power of the chemical system, and by ensuring that both are consistent we can ensure greater fidelity of molecular computing. We mention two instances where our solver verified such consistency. One is a "counter" tile assembly system in the aTAM model introduced in prior work that we verified also works in the TBN model [9]. Similarly, Section 3 shows an example of a strand displacement AND gate introduced in prior work that our solver verified to be correct in the TBN model as well [23].

Although we show that NP-hard problems can be encoded in TBNs, we in no way claim that TBNs by themselves provide an effective physical mechanism for solving NP-hard problems. Indeed, in general there are no guarantees on the time required to approach thermodynamically favored states.

2 Model

TBNs consider chemical systems affected by two (often opposing) thermodynamic driving forces: the free-energy benefit due to forming additional bonds, and the penalty for separate molecules joining together (free-energy of association). TBNs focus on these two driving forces due to their wide applicability, as well as the existence of systematic ways to amplify their strength relative to other driving forces [9]. A further aspect of the model that makes it inclusive is that it does not rely on geometric constraints to enforce correct behavior. Thus, in the TBN model, we think of a monomer as simply being an unstructured collection of binding sites.

Formally, a TBN is a triple $(D, ^*, \mathcal{T})$ where

- D is a finite set, which we call the set of *site type*s. These represent specific binding motifs, with the prototypical example of DNA "domains"—sequences that are designed to bind as a unit.
- $^*\colon D \to D$ is an involution (its own inverse) with no fixed point ($a^* \neq a$ for all a). This way the *complement* of the site type a^* is $(a^*)^* = a$. The prototypical example is that of DNA, where two sequences are complementary if they are Watson-Crick complements of each other.

– $\mathcal{T}$ is a finite multiset of monomer types, and a *monomer type* is a finite multiset over D. This models that a chemical systems consists of macromolecules and that macromolecules may have multiple binding sites.

We often call $\mathcal{T}$ alone a TBN when D and * are to be inferred. For example, given the following multiset of monomer types:

$$\mathcal{T}_{\text{ex}} = \{\{a, b^*\}, \{a, b^*\}, \{a^*, a^*, b\}\}$$

we can infer that $D = \{a, b, a^*, b^*\}$ and that * maps a to a^*, b to b^* and vice versa. We call an instance of a site type a *site*, and an instance of a monomer type a *monomer*. So $\mathcal{T}_{\text{ex}}$ has seven sites but just four site types, and it has three monomers but just two monomer types. We use a bold variable like $\mathbf{p}$ to indicate a monomer and an itilic variable like s to indicate a site.

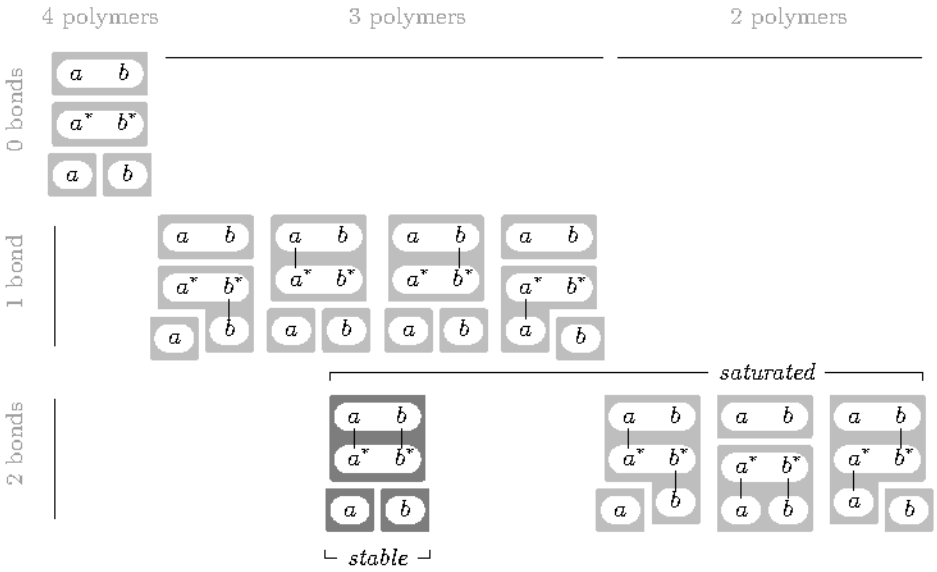

Fig. 1. All nine configurations of the TBN $\mathcal{T} = \{\{a\}, \{b\}, \{a, b\}, \{a^*, b^*\}\}$. We can infer the set of site types to be $D = \{a, b, a^*, b^*\}$, and we can infer that * maps a to a^* and b to b^* and vice versa. An edge between sites indicates that they pair in that configuration. A shaded box indicates a polymer.

As Figure 1 illustrates, a *configuration* γ of a TBN is a matching among its complementary sites. Two sites *pair* in γ if they are matched. Two monomers *bind* in γ if some of their sites pair. A *polymer* of γ is a connected component with respect to binding. The configuration γ is *saturated* if the matching is maximal. γ is *stable* if it is saturated and no saturated configuration has more polymers.

All else being equal, a state with more bonds formed is more favorable, and a state with more separate polymers is more favorable.[3] Thus "thermodynamic stability" is equated with "maximizing the number of molecular bonds and the number of separate polymers" subject to some prescribed trade-off between the two. Although the general case of quantitative trade-off is complex, TBNs take the limiting case in which bond strength is infinitely more favorable than configurational entropy. Importantly, many systems studied in molecular programming operate in this limit: in particular, DNA systems with long domains (strong binding sites) at relatively low concentrations [23].

3 Examples

The following examples illustrate the kinds of analysis the TBN model allows, motivated by the verification of different chemical systems.

3.1 Stable self-assembly

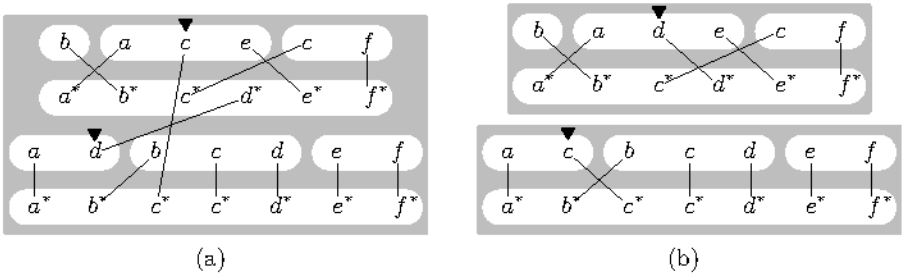

Fig. 2. (a) A stable configuration of a TBN $\mathcal{T}_a$ with a single polymer. Every stable configuration of $\mathcal{T}_a$ has a single polymer. (b) A stable configuration of another TBN $\mathcal{T}_b$. The only difference between $\mathcal{T}_a$ and $\mathcal{T}_b$ is the two sites marked by triangles swapping. With this subtle difference, $\mathcal{T}_a$ can fall apart into two polymers.

A core part of chemical and molecular machinery is assembling stable molecular structures. For example, biology relies on molecular structures like motors and capsids. Similarly, in engineering, molecular self-assembly techniques design building blocks that accrete into large structures. A basic question of self-assembly is whether a desired structure will hold together and how we can know.

Distinguishing stable from unstable assemblies can be challenging. Figure 2 shows two TBNs, each consisting of nearly the same collection of monomers, but there is a sharp difference in their behavior. One may fall apart into two

[3] Intuitively, the entropic benefit is due to additional microstates, each describing the three-dimensional position of each polymer, associated with such a state. See [9] for additional physical intution.

polymers in a stable configuration, while every stable configuration of the other holds together as a single polymer. How can we tell the two cases apart without enumerating the exponentially many configurations?

3.2 DNA strand displacement

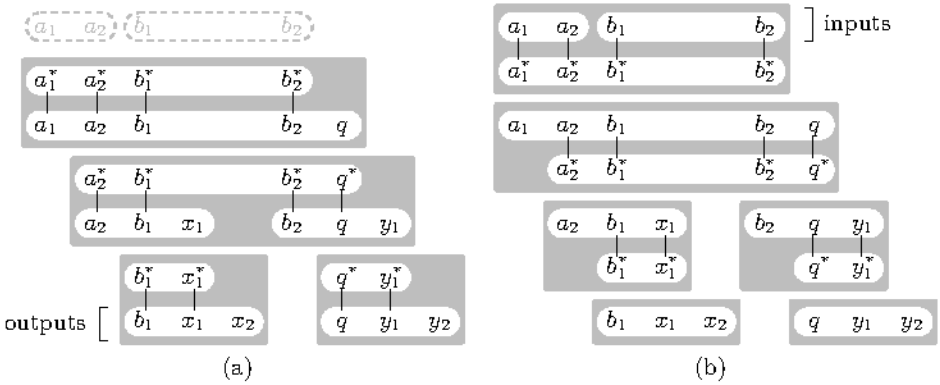

Fig. 3. (a) A TBN for a strand displacement cascade module that implements the abstract chemical reaction $A + B \rightarrow X + Y$ [23]. The input monomers are absent, and this is the unique stable configuration. The output monomers remain bound.[5] (b) The same TBN with the input monomers added. The configuration illustrated now becomes an additional stable configuration, and it has the output monomers free. A kinetic pathway that leads from the inputs being present to the outputs being free would correspond to flipping the bonds in a cascade from top to bottom.

Besides questions of assembly, TBNs can establish the correctness of chemical computation, in which signals are carried by monomers. In biology, chains of chemical reactions form intricate signaling pathways. In engineering, molecular circuits composed of chemical reaction cascades compute output signals from input signals.

Specifically, in a strand displacement cascade, computation is carried out by combinations of DNA strands [24]. When certain input strands are present, they can bind to DNA complexes and displace strands of the complex. These newly displaced strands act as input strands for other complexes. The cascade of displacements that follows can free certain output strands. Which are free encodes the output of the computation.

We can use a TBN to confirm that the output strands will not be released unless the correct input is present. Figure 3 shows two stable configurations of a TBN that models a strand displacement cascade. This construction was introduced

[5] The original system included domains known as toeholds. They do not appear here because in the TBN model we omit domains that bind weakly.

in prior work, but the TBN model gives us a way of precisely articulating the question and proving the correctness of the construction [23].

In the remainder of the paper, we formally define and resolve questions about TBNs such as these. In Sections 4 and 6 we focus on analyzing their computational complexity. In Section 5 we develop a method to answer them via a reduction to SAT. The tool we develop also handles the examples in this section and confirms the answers we have discussed here.

4 SaturatedConfig

Stable configurations are of central interest, so we need to be able to identify them. More generally, we ask what is the maximum number of separate polymers achievable in a saturated configuration of a TBN, which we call $S(\mathcal{T})$. Knowing this quantity, we can determine if a given configuration is stable (does it have $S(\mathcal{T})$ polymers?). We will see in later sections that other questions about TBNs that can be reduced to calculating $S(\mathcal{T})$ as well.

Definition 1. *$(\mathcal{T}, k)$ is in SaturatedConfig if some saturated configuration of the TBN $\mathcal{T}$ has at least k polymers. $S(\mathcal{T})$ is the greatest such k.*

In this section, we prove that SaturatedConfig is NP-complete and even that S is hard to approximate. To formalize our analysis, we establish the encoding of a TBN as follows. First, the encoding of a monomer is a sequence of its sites (arbitrarily ordered). Then the encoding of a TBN is a sequence of the encoding of its monomers (also arbitrarily ordered).

Claim 1. SaturatedConfig is in NP.

Proof. Consider a TBN $\mathcal{T}$ and an integer k. Suppose $\mathcal{T}$ has a saturated configuration γ with at least k polymers. Then γ is a certificate. Check that γ is saturated by checking that every two unpaired sites are not complements. Check that γ has k polymers by counting the connected components. □

To show hardness we reduce from the decision problem ExactCover, one of Karp's original NP-complete problems.

Definition 2. *A set X of sets is in ExactCover if some subset of X partitions $U = \bigcup X$. Such a subset is called an* exact cover*.*

The reduction relies on a transformation from an instance X of ExactCover to a TBN $\mathcal{T}(X)$, which Figure 4 illustrates. Let $U = \bigcup X$ be the set of elements of ..., and let $B = \sum X$ be the multiset of ... . Then $\mathcal{T}(X)$ has three kinds of monomers: $\mathbf{p}_x = x$ for each set x in X, $\mathbf{p}_U = \{s^* : s \in U\}$, and $\mathbf{p}_B = \{s^* : s \in B - U\}$. The number of polymers in a stable configuration of $\mathcal{T}(X)$ depends on whether X is a yes or no instance.

Claim 2.

$$S(\mathcal{T}(X)) = \begin{cases} 2 & \text{if } X \text{ is in ExactCover} \\ 1 & \text{otherwise} \end{cases}$$

	Include the monomer	
Due to	in $\mathcal{T}(X)$	in $\mathcal{T}_3(X)$
X	$\{a, b\}$	$\{a_1, b_1\}$
		$\{a_2, b_2\}$
	$\{b, c\}$	$\{b_1, c_1\}$
		$\{b_2, c_2\}$
	$\{c\}$	$\{c_1\}$
		$\{c_2\}$
Y	$\{a^*, b^*, c^*\}$	$\{a_1^*, b_1^*, c_1^*\}$
		$\{a_2^*, b_2^*, c_2^*\}$
$X' - Y$	$\{b^*, c^*\}$	$\{b_1^*, c_1^*, b_2^*, c_2^*\}$

(a)

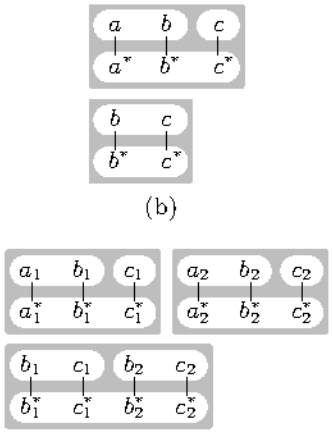

Fig. 4. (a) The transformation of the instance $X = \{\{a, b\}, \{b, c\}, \{c\}\}$ of ExactCover into the TBNs $\mathcal{T}(X)$ and $\mathcal{T}_3(X)$. The reason to include each monomer appears in the first column. (b) A saturated configuration of $\mathcal{T}(X)$ with two polymers, showing that X has an exact cover $\{\{a, b\}, \{c\}\}$. (c) A saturated configuration of $\mathcal{T}_3(X)$ with three polymers showing the same exact cover.

Proof. The $\mathbf{p}_x$ together perfectly complement $\mathbf{p}_U$ and $\mathbf{p}_B$ together. So in a saturated configuration γ, each $\mathbf{p}_x$ is completely bound to one or both of $\mathbf{p}_U$ and $\mathbf{p}_B$. So γ has one or two polymers.

Suppose X has an exact cover C. Then C partitions U. So bind $\mathbf{p}_x$ to $\mathbf{p}_U$ for all x in C. And bind $\mathbf{p}_x$ to $\mathbf{p}_B$ for all x not in C. This produces a saturated configuration with two polymers.

Conversely, suppose γ is a saturated configuration with two polymers. Then each $\mathbf{p}_x$ is completely bound to $\mathbf{p}_U$ or completely bound to $\mathbf{p}_B$. Those $\mathbf{p}_x$ bound to $\mathbf{p}_U$ then partition U and constitute an exact cover. □

This is enough to prove that SaturatedConfig is NP-complete, but to prove a strong hardness of approximation result for S, we scale the gap between the two cases up to a factor of j. To do so, we transform X into $\mathcal{T}_j(X)$ as Figure 4 illustrates. We combine $j - 1$ indexed copies of $\mathcal{T}(X)$ into a single TBN and then merge the $j - 1$ indexed copies of $\mathbf{p}_B$ into a single monomer. This way $\mathcal{T}(X)$ is the same as $\mathcal{T}_2(X)$, and in general we have the following.

Claim 3.

$$S(\mathcal{T}_j(X)) = \begin{cases} j & \text{if } X \text{ is in ExactCover} \\ 1 & \text{otherwise} \end{cases}$$

Proof (sketch). Bonds form only within, not between, copies in $\mathcal{T}_j(X)$. □

We now see that S is hard to approximate.

Definition 3. *A is a ρ factor approximation for f if $f(x)/\rho \leq A(x) \leq f(x)$ for all x.*

A trivial n factor approximation algorithm for S simply returns the number 1. This trivial algorithm turns out to be optimal.

Claim 4. No n^δ factor approximation algorithm for S runs in time polynomial in n for any $\delta < 1$ unless P = NP, where n is the number of monomers.

Proof. To the contrary, suppose A is such an algorithm, and consider a set X of m sets. Then $\mathcal{T}_j(X)$ has $n = (m+1)(j-1)+1 < 2mj$ monomers. So choose $j > (2m)^{\frac{\delta}{1-\delta}}$. Raise both sides to $1-\delta$ and rearrange to see that $j > (2mj)^\delta > n^\delta$. If X is not in ExactCover, then $S(\mathcal{T}_j(X)) = 1$. So

$$A(\mathcal{T}_j(X)) \leq S(\mathcal{T}_j(X)) = 1.$$

If X is in ExactCover, then $S(\mathcal{T}_j(X)) = j$. So

$$A(\mathcal{T}_j(X)) \geq S(\mathcal{T}_j(X))/n^\delta = j/n^\delta > 1.$$

So X is in ExactCover if and only if $A(\mathcal{T}_j(X)) > 1$. □

5 Computing SaturatedConfig

A hallmark property of an NP-complete decision problem is that some instances will be hard to solve unless P = NP. However, there are still interesting instances that can be solved efficiently. Since many real-world problems are NP-complete, various approaches have been developed to perform well on interesting instances. In this section, we apply such an approach. We encode the SaturatedConfig problem into SAT, the Boolean Satisfiability problem, which allows a SAT solver to find a solution that we can decode to obtain a saturated configuration.

5.1 The Boolean Satisfiability Problem

Definition 4. *A formula ϕ in propositional logic is called* satisfiable *iff there exists an assignment to the Boolean variables in ϕ such that the formula evaluates to true. The Boolean Satisfiability problem asks whether a given formula ϕ is satisfiable.*

A SAT solver is a tool that determines whether a formula has a satisfying assignment. In the last two decades, SAT solvers have become powerful enough to efficiently solve interesting instances of hard problems. The approach has been successful in areas such as hardware and software verification [6, 7, 13].

To use this approach to solve SaturatedConfig, we will translate a TBN $\mathcal{T}$ and an integer k into a CNF formula ϕ such that satisfying assignments of ϕ correspond to saturated configurations of $\mathcal{T}$ with at least k polymers. Recall that a CNF (conjunctive normal form) formula is a conjunction of disjunctions, such as $(x \vee \neg y) \wedge (x \vee z) \wedge z$.

5.2 Encoding saturated configurations

We construct a formula where satisfying assignments correspond to saturated configurations of a TBN $\mathcal{T}$. The formula uses a Boolean variable $\text{Pair}(s,t)$ for each pair of complementary sites s and t. Assigning $\text{Pair}(s,t)$ true will mean that s and t are paired. Otherwise they are unpaired. Note that pairing is symmetric, so $\text{Pair}(s,t)$ and $\text{Pair}(t,s)$ are the same variable.

In order to encode a valid configuration, we add the constraint

$$\leq_1 \{\text{Pair}(s,t) : t \in C(s)\}$$

to the formula for each site s, where $C(s)$ is the sites in $\mathcal{T}$ complementary to s. The direct encoding of $\leq_1$ (read "at most one") includes a binary clause $\neg\,\text{Pair}(s,t) \vee \neg\,\text{Pair}(s,u)$ for each two sites t and u in $C(s)$. The number of such clauses is quadratic in the size of $C(s)$. Although there do exist encodings that consist of only a linear number of binary clauses [20], due to the limited size of $C(s)$ in our test suite we found that the direct encoding works best.

Definition 5. *A site type a of a TBN $\mathcal{T}$ is* limiting *iff a^* occurs at least as many times as a in $\mathcal{T}$.*

In our example T_{ex}, b is the only limiting site type.

Claim 5. A configuration is saturated iff every limiting site is paired.

We omit the proof, but this fact allows us to easily encode saturation. To do so, we add the constraint

$$\bigwedge_{s \in L} \geq_1 \{\text{Pair}(s,t) : t \in C(s)\},$$

to the formula where L is the set of limiting sites of $\mathcal{T}$. Notice that each $\geq_1$ (read "at least one") constraint is simply a clause.

5.3 Encoding polymers

To begin identifying polymers, we convert site pairing to monomer binding using Boolean variables like $\text{Bind}(\mathbf{p},\mathbf{q})$. For now, assigning $\text{Bind}(\mathbf{p},\mathbf{q})$ true will mean that monomers $\mathbf{p}$ and $\mathbf{q}$ are bound. Note that binding is also symmetric, so $\text{Bind}(\mathbf{p},\mathbf{q})$ and $\text{Bind}(\mathbf{q},\mathbf{p})$ are the same variable. To convert pairing to binding, we add the constraint

$$\text{Pair}(s,t) \rightarrow \text{Bind}(\mathbf{p},\mathbf{q})$$

to the formula for each site s in a monomer $\mathbf{p}$ and each complementary site t in a different monomer $\mathbf{q}$.

We expand $\text{Bind}(\cdot,\cdot)$ to be transitive by adding the constraint

$$\text{Bind}(\mathbf{p},\mathbf{q}) \wedge \text{Bind}(\mathbf{p},\mathbf{r}) \rightarrow \text{Bind}(\mathbf{q},\mathbf{r})$$

to the formula for every three distinct monomers $\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$. The addition of transitivity ensures that monomers $\mathbf{p}$ and $\mathbf{q}$ are part of the same polymer iff $\text{Bind}(\mathbf{p},\mathbf{q})$ is true.

5.4 Maximizing the number of polymers

The most involved part of the encoding is enforcing the configuration to have at least k polymers. There are various ways to encode such a cardinality constraint, and the quality of that encoding can make or break our entire approach. Several techniques have been developed to automatically improve a low quality encoding [10, 16]. However, it typically pays to manually optimize it. We implemented several encodings, and describe here the one that resulted in the best performance, which was inspired by the representative encoding [12].

Let $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ be an arbitrary ordering of the n monomers of $\mathcal{T}$. We call a monomer $\mathbf{p}$ the *representative* of a polymer P iff every other monomer in P follows $\mathbf{p}$ in the ordering. To determine the representatives, we add the constraint

$$\mathrm{Bind}(\mathbf{p}, \mathbf{q}) \to \neg\, \mathrm{Rep}(\mathbf{q}).$$

to the formula for each monomer $\mathbf{p}$ that precedes each monomer $\mathbf{q}$.

Now, we can use these Boolean variables to enforce that a configuration has at least k polymers by adding

$$\geq_k \{\mathrm{Rep}(\mathbf{p}) : \mathbf{p} \in \mathcal{T}\}$$

to the formula. To effectively encode this cardinality constraint, we introduce new Boolean variables $\mathrm{Sum}(i, j)$ for $1 \leq i \leq n$ and $1 \leq j \leq k$, which will mean that among $\mathbf{p}_1, \ldots, \mathbf{p}_i$ there are at least j representatives. The encoding, which is a simplification of [20], is shown below and also illustrated in Figure 5.

$$\begin{aligned} \neg\, \mathrm{Sum}(i, j) &\to \neg\, \mathrm{Sum}(i+1, j+1) \\ \neg\, \mathrm{Sum}(i, j) \wedge \mathrm{Sum}(i+1, j) &\to \mathrm{Rep}(\mathbf{p}_{i+1}) \\ \neg\, \mathrm{Sum}(1, 2) & \\ \mathrm{Sum}(n, k). & \end{aligned}$$

Overall, the size of the formula we generate is bounded in terms of the number n of sites in a TBN. $O(n^2)$ clauses of size $O(n)$ encode a saturated configuration. $O(n^3)$ clauses of size $O(1)$ encode polymers, and $O(nk)$ clauses of size $O(1)$ check for k polymers. This gives us a total of $O(n^3)$ clauses of size $O(n)$ (since $k \leq n$).

6 Stably free

We will want to compute more complicated properties of stable configurations than just the number of polymers.

Definition 6. *A monomer is* free *in a configuration if no other monomer binds to it. A monomer* can be free *(*can be stably free*) in a TBN if it is free in some saturated (stable) configuration.*

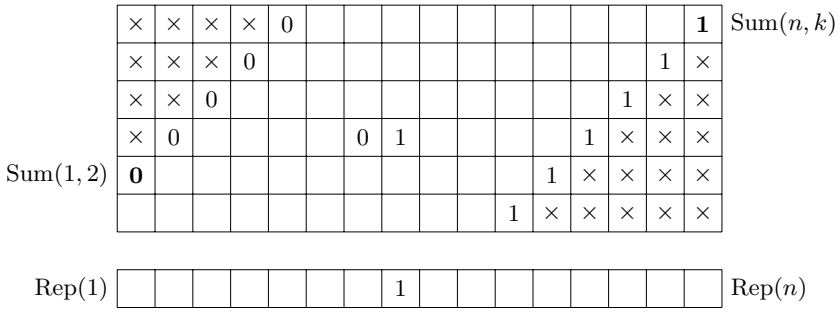

Fig. 5. An illustration of the Sum$(\cdot,\cdot)$ variables (above) and the Rep$(\cdot)$ variables (below) for a SaturatedConfig problem with $n = 16$ monomers and $k = 6$. Variables that are assigned to false/true are shown as 0/1. The zeros propagate up diagonally, while the ones propagate down diagonally. A 0,1 pair in a row (above) implies that the monomer at the position of the 1 is a representative (below). The bold entries indicate the unit clauses, while $\times$ indicates out redundant variables.

In a TBN that implements a computation, whether a certain monomer can be stably free can be interpreted as a bit of output [9]. The accompanying software package [1] shows two examples of AND gates from prior work that work this way [9]. The designated output monomer can be stably free iff both designated input monomers are present. Predicting such behavior is important to understanding the computations TBNs implement.

Definition 7. $(\mathcal{T}, \mathbf{p})$ *is in* StablyFree *if* $\mathbf{p}$ *can be stably free in* $\mathcal{T}$*.*

StablyFree seems harder than NP. For SaturatedConfig, a saturated configuration with k polymers served as a certificate. For StablyFree, a stable configuration with $\mathbf{p}$ free might seem to serve as a certificate. But checking that a configuration is saturated is easy, while checking that it is stable seems hard. So we look to a class larger than NP.

P^{NP} is the class of problems decided by some deterministic polynomial-time Turing machine with access to an oracle in NP. The Turing machine can alternate between computing and querying its oracle as it chooses. If we instead require the machine to make all of its queries in parallel as a single group, then we get the class $\mathrm{P}_{\parallel}^{\mathrm{NP}}$ [21].

Claim 6. StablyFree is in $\mathrm{P}_{\parallel}^{\mathrm{NP}}$.

Proof. Consider a TBN $\mathcal{T}$ and monomer $\mathbf{p}$. Let G_k (*r.* F_k) mean that some saturated configuration of $\mathcal{T}$ has k polymers (*r.* and has $\mathbf{p}$ free). Query G_k and F_k for each k from 1 to the number of monomers in $\mathcal{T}$ in parallel. Then the largest k that makes G_k true is $S(\mathcal{T})$. And $\mathbf{p}$ can be stably free iff $F_{S(\mathcal{T})}$. □

6.1 Hardness

To show hardness, we reduce from a graph problem. Recall that I is an independent set of a graph G if no two vertices in I are neighbors in G. I is maxi*mal* if no independent set properly contains it. I is maxi*mum* if no independent set has more vertices. For problems X and Y, we use the notation $\mathsf{X} \leq \mathsf{Y}$ to indicate that X can be reduced to Y via a polynomial time many-one reduction.

Definition 8. (G, v) *is in* MaxIndSet(Not)Member *if some maximum independent set of the graph G (does not) contain the vertex v.*

Claim 7. MaxIndSetMember $\leq$ StablyFree.

Proof. Consider an instance of MaxIndSetMember, that is, a graph G with n vertices and a particular vertex m. Construct a TBN $\mathcal{T}$ with $1 + n$ monomers as follows. Include a monomer $\mathbf{p} = E(G)$ consisting of the edges of G. Then for each vertex v of G, include a monomer $\mathbf{q}_v = \{e^* : v \text{ is incident to } e \in E(G)\}$ consisting of e^* for each edge e incident to v. Notice that the $\mathbf{q}_v$ can bind with $\mathbf{p}$ but not with each other, which makes $\mathbf{p}$ a sort of template.

Suppose G has an independent set I. Since the complement of I is a vertex cover, binding $\mathbf{q}_v$ to $\mathbf{p}$ for each v not in I yields a saturated configuration. It has $\mathbf{q}_v$ free for each v in I. Its number of polymers is the size of I plus one.

Conversely, suppose $\mathcal{T}$ has a saturated configuration γ. Then collecting each vertex v where $\mathbf{q}_v$ is bound to $\mathbf{p}$ yields a cover. The complement, each vertex v where $\mathbf{q}_v$ is free, constitutes an independent set. It contains v for any $\mathbf{q}_v$ that is free. Its size is the number of polymers in γ minus one.

So G has an independent set of size k containing m iff T has a saturated configuration with $k + 1$ polymers and with $\mathbf{q}_m$ free. □

We now connect MaxIndSetMember to MaxIndSetNotMember. First notice that, for example, the graph consisting of a single edge between vertices u and v has v in the maximum independent set $\{v\}$ and *not* in the maximum independent set $\{u\}$. So the two problems are not complements. We connect them in another way.

Claim 8. MaxIndSetNotMember $\leq$ MaxIndSetMember.

Proof (of Claim 8). Consider a graph G and a target vertex m. Form G' by first splitting each vertex. This means replace each vertex u with $\dot{u}$ and $\ddot{u}$ and replace each edge u–v with $\dot{u}$–$\dot{v}$, $\ddot{u}$–$\ddot{v}$, $\dot{u}$–$\ddot{v}$, $\ddot{u}$–$\dot{v}$. This way $\dot{u}$ and $\ddot{u}$ have the same neighbors. Then connect a new vertex m' to $\dot{m}$ and $\ddot{m}$. We will show that (G, m) is in MaxIndSetNotMember iff (G', m') is in MaxIndSetMember.

Let $[H]$ denote the maxi*mal* independent sets of the graph H. We claim that the mapping $f : [G] \to [G']$ is a bijection when defined by

$$f(I) = \begin{cases} \dot{I} \cup \ddot{I} & \text{if } m \in I \\ \dot{I} \cup \ddot{I} \cup \{m'\} & \text{else.} \end{cases}$$

To see that f is onto, consider I' in $[G']$. If $\dot{u}$ is in I', then by independence, no neighbor of $\dot{u}$ is in I'. By construction, these are also the neighbors of $\ddot{u}$. So by maximality, $\ddot{u}$ is in I'. So $I' \setminus \{m'\} = \dot{I} \cup \ddot{I}$ for some I. This way m is in I iff $\dot{m}$ and $\ddot{m}$ are in I'. By independence and maximality, this is iff m' is not in I'. So $I' = f(I)$. If we could add some u to I, then we could add $\dot{u}$ to I', but I' is maximal. So I is in $[G]$.

Now notice that $|f(I)|$ is $2\,|I|$ or $2\,|I| + 1$. So $|I| < |J|$ if and only if $|f(I)| < |f(J)|$. So I is a maxi*mum* independent set if and only if $f(I)$ is. And m is not in I if and only if m' is in $f(I)$. □

Now MaxIndSetNotMember is the same problem as MinVertexCoverMember, and MinVertexCoverMember has been shown to be $\mathrm{P}_{\|}^{\mathrm{NP}}$-hard [21]. So overall, StablyFree is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete.

Many other interesting properties of stable configurations are probably also $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete to decide. In particular, a monomer being stably free can be used as a signal to control the stable configurations of another TBN. The problem of deciding a property that can be controlled in this way will be as hard as StablyFree, which we would see by reducing from StablyFree. For instance, prior work introduces a weak-to-strong module $\mathcal{U}$ [9]. It can cause a monomer $\mathbf{p}$ to be free in *every* stable configuration of $\mathcal{T} \cup \mathcal{U}$ iff $\mathbf{p}$ is free in some stable configuration of $\mathcal{T}$.

6.2 Computing StablyFree

To decide more complex properties of stable configurations, we can modify the SaturatedConfig solver by adding appropriate constraints to have it search for saturated configurations with that property. But for StablyFree, we can use the SaturatedConfig solver as a black box.

Definition 9. *$|\gamma|$ is the number of polymers in a configuration γ.*

Claim 9. For a monomer $\mathbf{p}$ in a TBN $\mathcal{T}$, the following are equivalent.

1. $\mathbf{p}$ can be stably free in $\mathcal{T}$.
2. $\mathbf{p}$ can be free in $\mathcal{T}$ and $S(\mathcal{T} \setminus \{\mathbf{p}\}) \geq S(\mathcal{T}) - 1$.

Proof. For convenience, let $\mathcal{T}' = \mathcal{T} \setminus \{\mathbf{p}\}$.

Suppose $\mathbf{p}$ is free in a stable configuration γ of $\mathcal{T}$. Then $\mathbf{p}$ can be free in $\mathcal{T}$. Also, form the configuration γ' of $\mathcal{T}'$ by removing $\mathbf{p}$ from γ. Since $\mathbf{p}$ is free, γ' remains saturated. So $S(\mathcal{T}') \geq |\gamma'| = |\gamma| - 1 = S(\mathcal{T}) - 1$.

Conversely, suppose $\mathbf{p}$ can be free, but not stably free, in $\mathcal{T}$, and consider a stable cofiguration γ' of $\mathcal{T}'$. Form the configuration γ of $\mathcal{T}$ by adding $\mathbf{p}$ to γ' as a free monomer. Since γ remains saturated, γ is not stable. So $|\gamma| < S(\mathcal{T})$, so $S(\mathcal{T}') = |\gamma'| = |\gamma| - 1 < S(\mathcal{T}) - 1$, and so $S(\mathcal{T}') \ngeq S(\mathcal{T}) - 1$. □

Note that $\mathbf{p}$ can be free in $\mathcal{T}$ if none of its sites are limiting, a property that is easy to check.

7 Conclusion

The TBN model brings the complementary perspective of thermodynamic equilibrium to diverse models of chemical computation. In applying it, we discover two computational problems, SaturatedConfig and StablyFree, at the core of any questions about the behavior of TBNs. We prove tight bounds on their hardness, and present algorithms and implementations for solvers effective in many cases in practice.

Restricting TBNs in certain ways could be a useful design strategy for engineering systems that are powerful yet easy to verify. There may be interesting classes of TBNs for which the problems we consider are provably easy. Such classes might arise from imposing a specific global or local property on the TBN.

Alternatively, rather than restricting the classes of TBNs, understanding the nature of composition of TBN modules could achieve complex yet easy to verify behavior. However, it is not clear in general how the stable configurations of $\mathcal{T}_1 \cup \mathcal{T}_2$ relate to those of $\mathcal{T}_1$ and $\mathcal{T}_2$. It appears hard to "isolate" distinct functional parts in TBNs because stability is a global property.

Rather than ensuring that only desired configurations are stable, in practice we may need to consider unstable configurations (in particular, "close to stable") and limit how many of them are undesirable. If there are too many they may end up occurring with non-negligible probability. To avoid this we need to solve a counting problem like how many configurations with a given level of stability have a given bad property. How hard is this counting problem exactly? For instance, is it in #P?

Computing with a TBN as presented involves only looking at the existence of certain stable configurations. Instead, we can imagine unifying this thermodynamic equilibrium perspective with a kinetic perspective entirely within the TBN model—by asking about which configurations can be reached without traversing thermodynamically unfavorable configurations [3]. The computational difficulty of this problem, and what algorithms solve it, remain to be answered.

Acknowledgements. KB and DS were supported by NSF grants CCF-1618895 and CCF-1652824. We thank Lakshmi Prakash for helpful discussions, and for developing the Mathematica interface to the SAT solver.

References

1. https://bitbucket.org/ksbtex/tbnsolverm/.
2. Robert D Barish, Rebecca Schulman, Paul WK Rothemund, and Erik Winfree. An information-bearing seed for nucleating algorithmic self-assembly. *Proceedings of the National Academy of Sciences*, 106(15):6054–6059, 2009.
3. Keenan Breik, Cameron Chalk, David Doty, David Haley, and David Soloveichik. Programming substrate-independent kinetic barriers with thermodynamic binding networks. In *Proceedings of the 16th International Conference on Computational Methods in Systems Biology (CMSB)*, 2018.

4. Kevin M Cherry and Lulu Qian. Scaling up molecular pattern recognition with dna-based winner-take-all neural networks. *Nature*, pages 10.1038/s41586–018–0289–6, 2018.
5. Barry A Cipra. The ising model is NP-complete. *SIAM News*, 33(6):1–3, 2000.
6. Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
7. Fady Copty, Limor Fix, Ranan Fraer, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Y. Vardi. Benefits of bounded model checking at an industrial setting. In *CAV*, pages 436–453. Springer, 2001.
8. David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, 2012.
9. 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: 23rd International Conference*, pages 249–266. Springer, 2017.
10. Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *SAT 2005*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.
11. William E Hart and Sorin Istrail. Robust proofs of NP-hardness for protein folding: general lattices and energy potentials. *Journal of Computational Biology*, 4(1):1–22, 1997.
12. Marijn Heule and Stefan Szeider. A SAT approach to clique-width. *ACM Trans. Comput. Log.*, 16(3):24:1–24:27, 2015.
13. Franjo Ivančić, Zijiang Yang, Malay K. Ganai, Aarti Gupta, and Pranav Ashar. Efficient SAT-based bounded model checking for software verification. *Theoretical Computer Science*, 404(3):256–274, 2008.
14. Natasha Jonoska, Gregory L McColm, and Ana Staninska. On stoichiometry for the assembly of flexible tile dna complexes. *Natural Computing*, 10(3):1121–1141, 2011.
15. Rune B Lyngsø and Christian NS Pedersen. RNA pseudoknot prediction in energy-based models. *Journal of computational biology*, 7(3-4):409–427, 2000.
16. Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of boolean formulas. In *Proceedings of Haifa Verification Conference 2012*, 2012.
17. Luvena L Ong, Nikita Hanikel, Omar K Yaghi, Casey Grun, Maximilian T Strauss, Patrick Bron, Josephine Lai-Kee-Him, Florian Schueder, Bei Wang, Pengfei Wang, et al. Programmable self-assembly of three-dimensional nanostructures from 10,000 unique components. *Nature*, 552(7683):72, 2017.
18. Andrew Phillips and Luca Cardelli. A programming language for composable DNA circuits. *Journal of the Royal Society Interface*, 6(Suppl 4):S419–S436, 2009.
19. Rebecca Schulman and Erik Winfree. Programmable control of nucleation for algorithmic self-assembly. *SIAM Journal on Computing*, 39(4):1581–1616, 2009.
20. Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming*, pages 827–831, 2005.
21. Holger Spakowski. *Completeness for parallel access to NP and counting class separations.* PhD thesis, 2005.
22. Niranjan Srinivas, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik. Enzyme-free nucleic acid dynamical systems. *Science*, 358, 2017.
23. Chris Thachuk, Erik Winfree, and David Soloveichik. Leakless DNA strand displacement systems. In *Proceedings of DNA Computing and Molecular Programming 21*, pages 133–153. Springer, 2015.

24. David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, 2011.