Time Complexity of Computation and Construction in the Chemical Reaction Network-Controlled Tile Assembly Model

Nicholas Schiefer and Erik Winfree(✉)

California Institute of Technology, Pasadena, CA 91125, USA
winfree@caltech.edu

Abstract. In isolation, chemical reaction networks and tile-based self-assembly are well-studied models of chemical computation. Previously, we introduced the chemical reaction network-controlled tile assembly model (CRN-TAM), in which a stochastic chemical reaction network can act as a non-local control and signalling system for tile-based assembly, and showed that the CRN-TAM can perform several tasks related to the simulation of Turing machines and construction of algorithmic shapes with lower space or program complexity than in either of its parent models. Here, we introduce a kinetic variant of the CRN-TAM and investigate the time complexity of computation and construction. We analyze the time complexity of decision problems in the CRN-TAM, and show that decidable languages can be decided as efficiently by CRN-TAM programs as by Turing machines. We also give a lower bound for the space-time complexity of CRN-TAM computation that rules out efficient parallel stack machines. We provide efficient parallel implementations of non-deterministic computations, showing among other things that CRN-TAM programs can decide languages in $\mathsf{NTIME}(f(n)) \cap \mathsf{coNTIME}(f(n))$ in $\mathcal{O}(f(n) + n + \log c)$ time with $1 - \exp(-c)$ probability, using volume exponential in n. Lastly, we provide basic mechanisms for parallel computations that share information and illustrate the limits of parallel computation in the CRN-TAM.

1 Introduction

Biological organisms create remarkably sophisticated structures through the interplay of genetically-encoded chemical reactions and molecular self-assembly. DNA nanotechnology is beginning to explore the analogous potential of information-based chemistry by developing programmable circuitry using DNA strand displacement cascades [7,20,21,25,35], programmable self-assembly using DNA tile systems [3,12,22,29,31], as well as systems that combine both dynamic circuitry and self-assembly processes [33,34]. Whereas there are well-developed theoretical models for dynamic chemical circuits [5,18,26,27] and tile self-assembly [9,23,28] within which questions about the algorithmic power and efficiency of such systems can be posed and answered, the interplay of chemical reaction and self-assembly processes has received relatively little theoretical attention.

Y. Rondelez and D. Woods (Eds.): DNA 2016, LNCS 9818, pp. 165–182, 2016.
DOI: 10.1007/978-3-319-43994-5_11

It has been understood for decades that enzymes acting on information-bearing polymers can in principle perform efficient Turing-universal computation [4,6,13] and recently plausible molecular implementations using DNA nanotechnology have been proposed [14,19], but these studies do not exploit the full power of two- or three-dimensional self-assembly, nor do they explicitly concern themselves with how chemical reaction network computation can direct the construction of complex structures. We recently introduced a theoretical model [24], called the Chemical Reaction Network-Controlled Tile Assembly Model (CRN-TAM) that integrates the formal chemical reaction network (CRN) model [26] and the abstract tile assembly model (aTAM) [23]. We proved that in this model, the interplay between chemical reactions and self-assembly enables more efficient computation (in terms of space used) and more efficient construction (in terms of program size and shape scale) than either of the previous models alone. However, some of our constructions—devised to facilitate the proofs—were obviously inefficient in terms of time; they failed to exploit the inherent parallelism of molecular systems. Here, our goal is to determine whether integrating chemical reaction dynamics and tile self-assembly also provides as dramatic an advantage in terms of speed, for both computation and construction.

Using our previous definition of the structure and semantics of the CRN-TAM, we formulate a kinetic model based on the Gillespie dynamics of stochastic chemical reaction networks. Through a natural notion of CRN-TAM composition, we introduce the notion of an efficient encoding of an input and a fixed CRN-TAM decider that decides strings in a language, leading to the notion of copy-tolerant CRN-TAM deciders that can be readily composed and operated in parallel without interference. With reference to our previous stack machine construction, we show that there are space-efficient CRN-TAM deciders that use as much volume as a Turing machine would use space. We also give a lower bound showing that there are no space-efficient and copy-tolerant CRN-TAM deciders.

Our main result and most significant technical contribution is Theorem 5, which shows that there are copy-tolerant tile-based CRN-TAM deciders, which we demonstrate using a sentinel process like the one used by Adleman et al. [1] for analyzing the time complexity of assembling squares in a variant of the aTAM. To show how these copy-tolerant CRN-TAM deciders can be used for efficient parallel computation, we give a randomized CRN-TAM program that generates all 2^k strings of length k efficiently and with exponentially small error. We then combine these results to show that CRN-TAM programs can efficiently simulate non-deterministic computations in parallel, allowing them to (probabilistically) decide problems in $\mathsf{NTIME}(f(n)) \cap \mathsf{coNTIME}(f(n))$ in nearly $\mathcal{O}(f(n))$ time. Lastly, we show how the copy-tolerant CRN-TAM decider can be extended to allow limited state sharing between computations executing in parallel and discuss some likely limitations of parallelism in the CRN-TAM.

2 Semantics and Kinetics of the CRN-TAM

We begin by briefly reviewing the fundamental definition of the CRN-TAM, which was defined in detail by Schiefer and Winfree [24].

Definition 1. *A* tile *is an oriented square with a bond on each of the north, east, south, and west sides. Each of these* bonds *is a distinct tuple* (ℓ, s)*, with a label* ℓ *drawn from some alphabet and non-negative integer strength* s*. Formally, a tile is a four-tuple* $(\boxed{t}) = (N, E, S, W)$ *of bonds for the north, east, south, and west sides, respectively. In this paper, tiles are denoted by symbols surrounded by boxes.*

As in the abstract tile assembly model, tiles can aggregate to form larger structures. These structures are held together by the bonds on the edges of the tiles; to join onto an assembly, every tile must be bound with at least a minimum binding strength, or *temperature*:

Definition 2. *An* assembly *is an aggregation of adjacent tiles; formally, an assembly composed of tiles from a set* T *is a function* $A : \mathbb{Z}^2 \to (T \cup \{\varepsilon\})$ *that assigns to each side of the 2D lattice a tile from* T*. If* $A(x, y) = \varepsilon$*, the site is* empty *or* unoccupied*. A function* A *is a valid assembly if and only if:*

– *The occupied sites of the assembly form a connected set.*
– *The origin* $(0, 0)$ *is occupied by a tile* $A(0, 0) \in T$*; this tile is called the* seed *of the assembly.*
– *The total binding strength of the tile at each non-empty site is at least the temperature* τ*.*

Assemblies are denoted by symbols surrounded by double boxes. For example, $\boxed{t}$ *is a tile, but* $\boxed{\boxed{A}}$ *is an assembly. The (infinite) set of all valid assemblies using tiles from* T *is denoted* M_T*. When clear from context,* $\boxed{\boxed{X}}$ *is an assembly containing only the tile* $\boxed{x}$ *as its seed.*

In the CRN-TAM, the notion of a tile and assembly are essentially equivalent to those in models derived from the abstract tile assembly model [2,17,23,28]. In models that include only tiles, a tile set T completely specifies the structural form of a tile-based program.

Definition 3. *A CRN-TAM* program *is a tuple* (S, T, R, τ, I) *consisting of:*

– *A finite set* S *of identified* signal *species. We also use a notational "empty" species* ε *and let* $S_\varepsilon = S \cup \{\varepsilon\}$*.*
– *A finite set* T *of tuples* $\left(\boxed{t}, t^*\right)$ *pairing tiles* $\boxed{t}$ *and their* removal signals t^**, where* $t^* \in S_\varepsilon$*. The same tile may appear at most once in* T*, i.e. it cannot have two different removal signals.*
– *A finite set* R *of reactions, each of which is one of the following types:*
  - *CRN reactions* $A + B \xrightarrow{k} C + D$*, for signals* $A, B, C, D \in S_\varepsilon$*.*
  - *Tile deletion reactions* $A + \boxed{t} \xrightarrow{k} C + D$*, for signals* $A, C, D \in S_\varepsilon$*.*

- *Tile creation reactions* $A + B \xrightarrow{k} \boxed{t} + C$ *or* $A + B \xrightarrow{k} \boxed{t} + \boxed{t'}$, *for signals* $A, B, C \in S_\varepsilon$ *and tiles* $\boxed{t}$ *and* $\boxed{t'}$.
- *Tile relabelling reactions* $A + \boxed{t} \xrightarrow{k} B + \boxed{t'}$ *for signals* $A, B \in S_\varepsilon$ *and tiles* $\boxed{t}$ *and* $\boxed{t'}$.
- *Tile activation reactions* $A + \boxed{x} \xrightarrow{k} \boxed{\boxed{X}} + x^*$ *where* $A \in S_\varepsilon$ *and* $\left(\boxed{x}, x^*\right) \in T$.
- *Tile deactivation reactions* $\boxed{\boxed{X}} + x^* \xrightarrow{k} A + \boxed{x}$ *where* $A \in S_\varepsilon$ *and* $\left(\boxed{x}, x^*\right) \in T$.

In all of these reactions, a reactant or product ε *indicates that the reactant or product does not exists; for example, a reaction* $A + \varepsilon \xrightarrow{k} \varepsilon + D$ *is just* $A \xrightarrow{k} D$. *The reaction rate constant* k *must be specified as a rational number.*

– *The temperature* $\tau \in \mathbb{N}$, *which is the minimum binding strength.*
– *The initial state* I, *a multiset of tiles and signals that are initially present. Often, we will use* I *as a function* $I : (S \cup T) \to \mathbb{N}$ *indicating the number of a particular element in the multiset. An initial state does not contain any assemblies.*

Structurally, the signal species in S are analogous to the species of a stochastic chemical reaction network (sCRN) and the tiles in T are analogous to the tile set that defines a program in the abstract tile assembly model (aTAM). Signal species and unbound tiles float in a well-mixed vessel, interacting analogously to the species in a stochastic chemical reaction network.

Occasionally, it will be useful to *combine* several CRN-TAM programs.

Definition 4. *The* combination *of two CRN-TAM programs* $P = (S, T, R, \tau, I)$ *and* $P' = (S', T', R', \tau, I')$ *is* $P \oplus P' = (S \cup S', T \cup T', R \cup R', \tau, I \cup I')$, *so long as tiles are consistent, i.e.,* $\left(\boxed{t}, x\right) \in T$ *and* $\left(\boxed{t}, y\right) \in T'$ *implies* $x = y$. *Note that since* I *and* I' *are multisets, duplicates are repeated in the union.*

As in the aTAM, tiles bind together to form assemblies provided that they attach with a total binding strength that is at least the temperature τ. Along with the reaction specified in R, there are implicit *addition and removal reactions* in the CRN-TAM that are similar to the corresponding reactions in other tile assembly models, but with a slight change in character; extended rationale is given in our previous paper introducing the CRN-TAM [24].

Definition 5. *Let* $(\boxed{t}, t^*) \in T$, *and let* $\boxed{\boxed{A}}$ *and* $\boxed{\boxed{B}}$ *be assemblies that differ by exactly* $\boxed{t}$ *in some location other than* $(0, 0)$. *A* tile addition reaction *is a reaction*

$$\boxed{\boxed{A}} + \boxed{t} \xrightarrow{1} \boxed{\boxed{B}} + t^*$$

Notice that since $\boxed{\boxed{B}}$ *is valid,* $\boxed{t}$ *is attached with total strength at least* τ*. The corresponding removal reaction*

$$\boxed{\boxed{B}} + t^* \xrightarrow{1} \boxed{\boxed{A}} + \boxed{t}$$

may occur only when $\boxed{t}$ *is bound by* exactly *strength* τ*, and the removal signal is not* ε*. Note that the seed tile is privileged and cannot be removed from the assembly; it may be deactivated only after all other tiles have been removed.*

The signals, free tiles, and assemblies in the reaction vessel completely specify the state of a CRN-TAM program at any time:

Definition 6. *A* state *L of a CRN-TAM program P is a multiset of signals, tiles, and assemblies. As with the initial state I, we use the notation $L(x) : (S \cup T \cup M_T) \rightarrow \mathbb{Z}^+$ to refer to the current count of x in L.*

Frequently, we will refer to *the program state* or *current state*, which is simply the state that reflects the current contents of the reaction vessel.

Definition 7. *A reaction is* possible *for a state L if its rate constant is nonzero and for every one of its reactants α, $L(\alpha) > 0$. The* possible reactions $\text{Pos}(L)$ *of a state L include all of the possible reactions in R and all of the possible tile addition and removal reactions. Note that* $\text{Pos}(L)$ *is always finite.*

The possible reactions induce a graph that describes the possible transitions between different states.

Definition 8. *The* reaction graph *$G(P)$ of a CRN-TAM program $P = (S, T, R, \tau, I)$ is a directed graph with a vertex for each of the (infinitely numerous) states of P and a directed edge from L to all states in* $\text{Pos}(L)$ *for all states L. The* reachable reaction graph *is a subgraph of $G(P)$ with only vertices that are descendants of the initial state I. Where it is unambiguous or unimportant, we may refer to the reachable reaction graph as simply the reaction graph.*

Note that reachable reaction graphs are by definition connected, but may not be strongly connected. As reactions occur, the program state will change from one state to the other in a manner that is governed by the reactions' propensities: in this paper, we are especially interested in CRN-TAM programs that eventually reach a state with no possible reactions, where they will remain forever.

Definition 9. *A state of a CRN-TAM program is a* termination state *if it has no possible reactions. Equivalently, the termination vertex has no out edges in the reaction graph. A CRN-TAM program P* stops *if it reaches a termination state with probability one and the set of reachable states is finite.*

As we will see in Definition 11, we cannot consider the temporal dynamics of a CRN-TAM program without knowing the volume in which the program operates. In this paper, we will always use a default volume, dependent on the program, that ensures that any execution path will at all times have a finite density, in the following sense:

Definition 10. *An* atomic chemical species *is a signal or a tile, whether free or bound to an assembly. The* mass *of a state is the total number of atomic chemical species present in all signals, tiles, and assemblies. The* volume *required by a CRN-TAM program P is the maximum mass present for any state in the reachable state graph of P.*

By this definition, if the reachable reaction graph is infinite, then it must have a state whose mass exceeds any given bound, and thus violates the finite density constraint for some possible execution path. Our choice of volume cannot handle such systems, and in this paper we restrict our attention only to CRN-TAM programs with finite reachable reaction graphs. In the CRN-TAM, as in stochastic chemical reaction networks [26], the volume is fixed at the beginning and does not change over time.

Thus far, we have specified only the possible reactions associated with a state and not the dynamics of the system. The program state evolves according to stochastic Gillespie dynamics, where reactions occur at a rate proportional to their current *propensity* [10,11].

Definition 11. *Let $P = (S, T, R, \tau, I)$ be a CRN-TAM program in state L that uses volume V. The* propensity *of a reaction R with rate constant k is given by:*

- *$\rho(R) = kL(R_1)$ if R is a unimolecular reaction with reactant R_1.*
- *$\rho(R) = kL(R_1)L(R_2)/V$ if R is a bimolecular reaction with two distinct reactants R_1 and R_2.*
- *$\rho(R) = kL(R_1)(L(R_1) - 1)/V$ if R is a bimolecular reaction with identical reactants.*

Note that a reaction is possible if and only if its propensity is nonzero. With reference to Definition 11, we can finally define the kinetics of a CRN-TAM program.

Definition 12. *Let $L(t)$ be a random variable-value for the current state of a CRN-TAM program P at a time $t \in [0, \infty)$. The state $L(t)$ evolves over time as a continuous time Markov chain on the space of possible CRN-TAM states with (deterministic) initial state $L(0)$. For two distinct program states A and B, the transition rate between them is given by the propensity of the reaction in P that converts state A into state B, 0 if there is no such reaction, or the sum of propensities if there is more than one (e.g. $X \xrightarrow{1} Y$ and $C + X \xrightarrow{1} C + Y$).*

For each of stochastic CRNs and the abstract tile assembly model, there is a natural notion of the "size" of a molecular program; in sCRNs, this is the number of reactions, while in the aTAM it is the number of tiles. We can similarly define a measure of program complexity for the CRN-TAM.

Definition 13. *The complexity of an initial state $I : (S \cup T) \to \mathbb{N}$ is*

$$|I| = \sum_{z \in (S \cup T)} \log_2(I(z) + 1)$$

This definition is natural since it is the number of bits needed to specify a general initial state I, up to small constant multiplicative and additive factors. We similarly define the size of a set of reactions such that if all reactions have unit rate constants, it is just the count of reactions, but otherwise it scales as the information needed to specify the rates:

Definition 14. *The complexity of a set of reactions R, where $r \in R$ is written as $A_r + B_r \xrightarrow{k_r} C_r + D_r$ and $k_r = \frac{n_r}{d_r}$ as an irreducible fraction, is*

$$|R| = \sum_{r \in R} \log_2(n_r \times d_r + 1)$$

The complexity of a CRN-TAM program is the sum of the complexities of its components.

Definition 15. *Let $P = (S, T, R, \tau, I)$ be a CRN-TAM program. The* complexity *of P with respect to temperature τ is*

$$\begin{aligned} K^{\tau}_{\mathrm{CT}}(P) &= |S| + |T| + |R| + |I| \\ &= |S| + |T| + \sum_{r \in R} \log_2(n_r \times d_r + 1) + \sum_{z \in (S \cup T)} \log_2(I(z) + 1) \end{aligned}$$

Each term is related to the number of bits required to specify the corresponding part of the program, up to logarithmic factors. Like sCRNs but unlike the aTAM, we allow nontrivial initial state as a convenience; our previous work showed that for any CRN-TAM program P, there is a CRN-TAM program P' with no initial state and program complexity $K^{\tau}_{\mathrm{CT}}(P') = \Theta(K^{\tau}_{\mathrm{CT}}(P))$ that simulates it [24, Theorem 4].

3 Efficient Computation

We are principally concerned with using CRN-TAM programs to perform efficient computation. As in much of theoretical computer science, we deal primarily with decision problems, and therefore formulate a model of computation that solves them. As defined so far, CRN-TAM programs cannot "compute" in the sense that a Turing machine or even a circuit can; a program is fixed and has no notion of input. Furthermore, most of the efficient and natural encodings of fixed strings in the CRN-TAM involve assemblies, while a CRN-TAM program does not have any assemblies in its initial state. To resolve this, we begin by describing a natural way of encoding fixed input strings as CRN-TAM programs, and then demonstrate how those input strings can be combined with CRN-TAM *deciders* to provide a model of computation.

Definition 16. *Consider a CRN-TAM program P and a string x over an alphabet Σ. Let $T_{\Sigma} : T \to \Sigma$ be a partial function from the tiles of P to the alphabet, assigning some of the tiles in T to represent symbols in the alphabet. We say that P encodes x if it constructs a $1 \times (|x| + 1)$ rectangular assembly A with the following properties:*

– *A begins with a designated "start tile"* $\boxed{t_{\text{start}}}$.
– *Let the* k^{th} *tile after the start tile be* $\boxed{t_k}$. *Then* $T_\Sigma\left(\boxed{t_k}\right) = x_k$, *the* k^{th} *symbol in the string.*

Of course, a string of length n can always be encoded by a CRN-TAM program with $\Theta(n)$ complexity, by using a distinct tile type for each symbol in the string. Previously, Adleman et al. [1] and Soloveichik and Winfree [28] showed that strings of length n can be encoded[1] in aTAM tile sets with $\mathcal{O}(n/\log n)$ distinct tile types at temperature $\tau = 2$. We can show that CRN-TAM programs with sublinear complexity can encode n-symbol strings. In [24] it was shown that:

Theorem 1. *For any string x over a constant-size alphabet Σ and temperature $\tau \geq 1$, there is a CRN-TAM program $P = (S, T, R, \tau, I)$ that encodes x and has complexity $K^\tau_{\text{CT}}(P) = \mathcal{O}(n/\log n)$.*

Moreover, this bound is tight. We therefore have a natural definition of input for computation with the CRN-TAM:

Definition 17. *An* input encoding *of a string x (over an constant alphabet) is a CRN-TAM that encodes x and has complexity $K^\tau_{\text{CT}}(P) = \mathcal{O}(|x|/\log|x|)$ at any temperature $\tau \geq 1$.*

Our CRN-TAM computers should be independent of the input and should solve an appropriate decision problem when combined with an input encoding. For the rest of the paper, we will use a default encoding $E_\Sigma(x)$ for alphabet Σ and input string x, which we will simply refer to as $E(x)$ where the alphabet is clear from context.

Definition 18. *Consider a language $\mathcal{L}$ that is decidable by a Turing machine M. Consider a fixed CRN-TAM program D with two identified signals Q_{accept} and Q_{reject}. We say that D is a* CRN-TAM decider *for $\mathcal{L}$ with respect to the default input encoding E if, for every input x the combined program $D \oplus E(x)$:*

– *Produces Q_{accept} and then stops immediately if and only if M accepts x.*
– *Produces Q_{reject} and then stops immediately if and only if M rejects x.*

For convenience, we will say that D accepts x if the first case holds and rejects x if the second case holds.

For any execution of a CRN-TAM decider acting on an input, there is a time t^* for which $L(t) = L(t^*)$ for all $t \geq t^*$. Using the dynamics described in Definition 12, we can use this to define the amount of time that a computation in the CRN-TAM takes.

[1] By necessity, a different notion of "encoding" must be used in the aTAM, since building even a $1 \times n$ rectangle requires $\Theta(n)$ tile types [2]. However, the notion used in the aTAM is analogous to our notion of encoding.

Definition 19. *Consider a CRN-TAM decider D for language $\mathcal{L}$, and a string x. The* time *that $D + E(x)$ takes to decide x is the random variable T^* for the minimum time t^* so that $L(t) = L(t^*)$ for all $t \geq t^*$. We call this the* stopping time *of the CRN-TAM program. As we are usually interested in asymptotically characterizing the worst-case time that a decider takes to decide all inputs of a given length, we define the random variable $T(n) = \max_{x \in \Sigma^n} T^*(x)$.*

The aTAM is an inherently scalable model of molecular computation, in the sense that we may consider an arbitrary number of assemblies growing in parallel and executing independent computations. Because there is no mechanism for inter-assembly interaction and the supply of tiles is fixed (and implicitly infinite), each assembly is a universe unto itself; from the perspective of an aTAM programmer, it does not matter if a reaction vessel has a single assembly or a million. Like stochastic CRNs, however, the CRN-TAM has a shared global state. As a result, a CRN-TAM program does not necessarily scale for parallel execution: combining just two functioning CRN-TAM programs may not produce a functioning CRN-TAM program. Nonetheless, scalability is a desirable property from a theoretical standpoint, since it might allow parallel computations, and from a practical standpoint, where any molecular program is unlikely to have an isolated reaction vessel. As a seemingly minimal base case, a CRN-TAM program that can scale for parallel execution ought to still work correctly when multiple copies of the same program act on the same input. We are therefore especially interested in CRN-TAM deciders that are copy-tolerant, in the following formal sense that is closely related to the similarly-named notions for CRNs [8,15]:

Definition 20. *A CRN-TAM decider D for language $\mathcal{L}$ is copy-tolerant if for any $k \geq 2$, $D'_k = \bigoplus_{i=1}^{k} D$ is also a CRN-TAM decider for $\mathcal{L}$.*

Intuitively, a copy-tolerant CRN-TAM decider is one that supports running multiple instances of the decider in the same reaction vessel simultaneously and still reports the answer accurately. As we will see, many convenient CRN-TAM deciders are not copy-tolerant, and there appear to be substantive lower bounds on the volume required by copy-tolerant CRN-TAM deciders.

4 Space-Efficient Deciders

Definition 21. *A site (i, j) containing tile $\boxed{x}$ is* immediately dependent *on site (i', j') containing tile $\boxed{x'}$ if it shares a bond with $\boxed{x'}$ and was added to the assembly after $\boxed{x'}$. The site (i, j) is* dependent *on (i', j') if it shares a bond with a tile that is dependent or immediately dependent on $\boxed{x'}$.*

The recursive definition of dependency induces a directed, acyclic graph of dependencies, where edges go from tiles to the tiles they are immediately dependent on. In such a graph, a tile's descendants are all the tiles it is dependent on, and a tile's ancestors are all those tiles that depend on it.

Definition 22. *Consider an assembly A. The* dependency graph *of A is a directed graph with the occupied sites of A as vertices and a directed edge from each site to the sites on which it is immediately dependent.*

As an immediate corollary of Definition 22, the dependency graph is acyclic and rooted at the seed of the assembly, as proved in [24]. Dependency is a critical concept for attempting to disassemble an assembly; to disassemble, we are forced to remove only leaves of the dependency DAG, performing the entire disassembly in "dependency-reversed" order.

Theorem 2. *A tile cannot be removed until all of its ancestors in the dependency graph have been removed. That is, a tile cannot be removed until all tiles at sites dependent on it have been removed.*

Theorem 2 has both reassuring and limiting consequences. On one hand, it ensures that tiles cannot be ripped out from the middle of the assembly when their removal signals are present. On the other, it implies that we can never create "temporary scaffolding" for our CRN-TAM constructions: a CRN-TAM program may never build parts of an assembly that are dependent on scaffolded parts that are meant to be removed later.

If a language $\mathcal{L}$ can be decided on a Turing machine using space $s(n)$, we might hope that there is a CRN-TAM decider for L that uses only $\Theta(s(n))$ volume. Indeed, the existence of such deciders is an immediate corollary of the stack machine construction from [24], itself a modification of the stack polymer construction by Qian et al. [19].

Theorem 3. *Given a language $\mathcal{L}$ decided by a Turing machine in space $s(n)$, there is a CRN-TAM decider for $\mathcal{L}$ that uses $\Theta(s(n))$ volume.*

The stack machine construction relied critically on storing the state of the stack machine in the global CRN and is therefore not copy-tolerant. We might hope to construct a CRN-TAM decider that is similarly space efficient, using asymptotically as much volume as a Turing machine uses space on its tape, but that also remains copy-tolerant. Unfortunately, this turns out to be impossible.

Theorem 4. *Consider a language $\mathcal{L}$ decidable by a Turing machine that requires $s(n)$ space and $t(n)$ time. Every copy-tolerant CRN-TAM decider for $\mathcal{L}$ uses volume $\Omega(t(n))$.*

Proof. Omitted due to lack of space. □

In general, there are many functions where $t(n) \in \omega(s(n))$, i.e. that require much more time to compute than space, and so there are languages for which space efficient, copy-tolerant CRN-TAM deciders do not exist.

5 Time Complexity of Tile Computations

Having established that stack machine-based CRN-TAM deciders experience a slowdown proportional to the volume and that copy-tolerant deciders require $\Omega(t(n))$ volume, we safely abandon attempts to build copy-tolerant stack machine-type deciders. If we have temperature at least 2, where cooperative binding is possible, we can also perform Turing-universal computation using tiles alone [23,30,32]. This approach proves to be very fruitful, giving us a key theorem:

Theorem 5. *For every language $\mathcal{L}$ decidable in time $t(n)$ and using space $s(n)$ on a Turing machine, there is a copy-tolerant CRN-TAM decider D for $\mathcal{L}$ with expected time complexity $\Theta(t(n))$ and volume complexity $\Theta(t(n)s(n))$.*

Proving Theorem 5 is somewhat technical and involves some careful stochastic analysis. The key problem is that CRN-TAM programs have only a finite supply of each tile type present at any given moment. In the aTAM and many of its derivatives, this is not a problem, since an infinite supply of tiles with fixed ratios is generally assumed. The issue is further complicated by the finite density constraint and the desire for *fast* computation; having all the necessary tiles present from the beginning would raise the volume prohibitively and make computation needlessly slow. To resolve this in the CRN-TAM, we use a simple mechanism for regenerating consumed tiles; although the mechanism is intuitive, the asynchronicity of efficient computation as well as fluctuations in the tile concentrations make it harder to analyze. Our proof that this mechanism leads to efficient computation—as our intuitions tell us—involves analyzing a *sentinel process* like the one introduced by Adleman et al. [1] in a variant of the aTAM, where we artificially constrain the dynamics to create a stochastic process that is easy to analyze but still stochastically dominates the actual CRN-TAM dynamics. We analyze the dynamics of the sentinel process as a phase-type distribution on CRN-TAM states. Lastly, we apply some careful combinatorics and the Chernoff bound for exponentially distributed random variables to show Theorem 5.

At a conceptual level, our efficient and copy-tolerant CRN-TAM decider will use a constant-sized Turing-universal aTAM tile set to perform the computation proper, by the well-known method of computation histories (Fig. 1). To ensure that the (bimolecular) tile addition reactions proceed quickly, the decider will use a tree-structured counter to efficiently generate an $\Theta(V)$ concentration of each of those (constant in number) tiles. Lastly, the tile removal signal will be consumed after its release and used to produce a new tile, which will replace the consumed tile in the "pool" of available tiles. Each of these conceptual points will be shown as a separate lemma, along with a number of technical lemmas used for the analysis of the sentinel process.

To begin, we adapt a classic result from the aTAM [30], to show that there are effective tile-only CRN-TAM deciders which make only minimal use of the CRN.

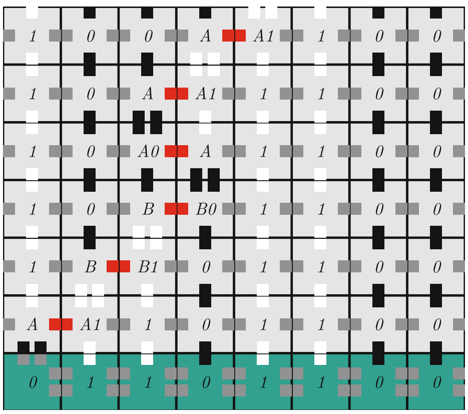

Fig. 1. An illustration of the assembly built by the tile-based CRN-TAM decider TuringTiles, in this case implementing a Turing machine with two states (A and B) operating on the binary alphabet $\Sigma = \{0, 1\}$ and the rule that the read head goes left whenever it sees a 0 and goes right whenever it sees a 1, flipping each bit as it goes. The input string (01101100) is shown at the bottom.

Lemma 1. *For any decidable language, there is a CRN-TAM program*

$$\mathsf{TuringTiles} = (\{Q_{\text{accept}}, Q_{\text{reject}}\}, T, 2, \varnothing)$$

where Q_{accept} is the removal signal for a tile t_{accept} that indicates acceptance and Q_{reject} is the removal signal for a tile t_{reject} that indicates rejection, consisting of only tiles and the necessary signals, that is a CRN-TAM decider when combined with an initial state containing a large enough supply of tiles.

Proof. Omitted due to lack of space. □

In the CRN-TAM, it is critically important to have a high concentration of tiles for addition since tile addition reactions are bimolecular. By definition, assemblies have unit concentration, so we must aim to have $\Theta(V)$ tiles of a given type in the reaction vessel of volume V in order to have constant expected time for tile addition. For the case considered by Theorem 5, however, at the beginning neither the input encoding nor the CRN-TAM decider contains enough tiles; thankfully, they can be generated efficiently.

Lemma 2. *Given a species (signal or tile) A, there is a CRN-TAM program $\mathsf{TreeCounter}_n(A)$ that stops with 2^n copies of A in $\mathcal{O}(n)$ expected time, with program complexity $K^1_{\text{CT}}(\mathsf{TreeCounter}_n(A)) \in \mathcal{O}(n)$.*

Proof (sketch). For every $0 \leq i \leq n$, introduce the signal S_i and the reaction $S_i \rightarrow S_{i+1} + S_{i+1}$ unless $i = n$, with S_0 as the sole species present in the initial state. Notice that the total number of S_i produced is precisely twice the number of S_{i-1} that were produced, so by induction a total of 2^n instances of S_n are produced by this program. By adding the reaction $S_n \to A$, we are guaranteed to eventually stop with precisely 2^n copies of A.

Through methods similar to those that will be used in the proof of Theorem 5, we can show that $\mathsf{TreeCounter}_n(A)$ stops in $\mathcal{O}(n)$ expected time. The key insight is that every reaction in the tree counter is unimolecular and so can always proceed at rate $\Omega(1)$. □

Lastly, there is a simple mechanism for regenerating tiles, which we name for convenience.

Definition 23. *For a tile* $(\boxed{t}, t^*)$ *in a CRN-TAM program, we define the program* $\mathsf{ReplaceTile}_\tau(\boxed{t}, t^*) = (S, T, R, \tau, I) = \Big(\{t^*\}, \{\big(\boxed{t}, t^*\big)\}, \{t^* \to \boxed{t}\}, \tau, \varnothing\Big)$.

Combining Lemmas 1 and 2, we can efficiently generate a $\Theta(V)$ concentration for all of the tiles in the (constant-sized) tile set for computation in only $\Theta(\log V)$ time; so long as the concentration remains that high, tile addition reactions will happen in constant expected time and each tile addition will release a removal signal t^* that will be converted back into an active tile in constant expected time. We now have the preliminaries necessary to state our construction of the CRN-TAM decider in Theorem 5:

Proof (construction for Theorem 5). Consider a CRN-TAM program $D = (S, T, R, 2, I)$ consisting of the combination of $\mathsf{TuringTiles}$ and, for all tiles $(\boxed{t}, t^*) \in T$, both $\mathsf{TreeCounter}_{\lceil \log V \rceil}(\boxed{t})$ and $\mathsf{ReplaceTile}_2(\boxed{t}, t^*)$. Since the tree counter will produce $\Theta(V)$ tiles of each tile type right from the beginning, the combined program is a CRN-TAM decider by Lemma 1. Lastly, notice that D is copy-tolerant, since the entirety of the computation happens on a single assembly. □

The analysis of the expected time for D to decide L, including several technical lemmas, will appear in the full paper.

6 Combinatorial Assembly Production and Nondeterministic Parallelism

The copy-tolerant CRN-TAM decider in Theorem 5 allows us to perform several threads of computation in parallel, given sufficient volume. In general, it is simple to make a copy of the input for each thread efficiently, using a tree-style copier like the one used in Lemma 2. However, identical inputs are not useful; any deterministic tile set such as the tile-based CRN-TAM decider would produce precisely the same computation in all of the parallel threads. To remedy this, we might hope to generate identifiers for each of the k threads as they are produced during the input copying process. To this end, we could perform a *combinatorial assembly* task, like generating all 2^k binary strings of length k in a serial fashion as was done in [24], but this would take time exponential in k. If we are willing to allow some small chance of error, a very simple CRN-TAM program can assemble these strings in $\Theta(k)$ time.

Theorem 6. *For any positive integer c, there is a CRN-TAM program* $\mathsf{CombinGenerate}(n)$ *with complexity* $K^1_{\mathrm{CT}}(\mathsf{CombinGenerate}(n)) \in \Theta(cn)$ *that stops having constructed all* 2^n *assemblies, each of size* $1 \times (n+1)$*, encoding all binary strings of length n, in time* $\Theta(\log(cn2^n))$ *with probability* $1 - e^{-c}$.

Proof. Omitted due to lack of space. □

We would of course prefer to be able to do combinatorial string assembly deterministically and efficiently. Intuitively, this seems extremely difficult, for the following reason. Suppose that we have assembled all but one of the strings; if we are operating with some kind of parallelism, how can we know which string has not yet assembled without some kind of exponential communication problem? While we have not been able to prove anything beyond a few special cases, we suspect efficient, deterministic combinatorial assembly is not possible.

Conjecture 1. There is no CRN-TAM program that constructs all 2^n binary strings of length n and runs in $\mathcal{O}(\mathsf{poly}(n))$ time.

As mentioned earlier, this combinatorial assembly can be used for generating various *seeds* for parallel computations, so that different parallel threads can operate differently. A simple application of this concept is implementing parallel non-determinism, where each seed acts as a string of binary guesses for a nondeterministic Turing machine.

Theorem 7. *For any language* $\mathcal{L} \in \mathsf{NTIME}(f(n)) \cap \mathsf{coNTIME}(f(n))$ *and positive integer c, there is a CRN-TAM decider for $\mathcal{L}$ that decides it in* $\Theta(f(n)+n+\log c)$ *expected time, with probability at least* $1 - e^{-c}$.

Proof (sketch). First, observe that an input encoding can be converted into one that generates 2^f copies of the input by replacing each production of every tile $\boxed{t}$ with an instance of $\mathsf{TreeCounter}_f(\boxed{t})$ instead. Per our analysis of the tree counter, this takes only $\mathcal{O}(f)$ time, so we can generate $cn2^n$ copies of the input in only $\mathcal{O}(n + \log c)$ time. Observe that a nondeterministic Turing machine running in $f(n)$ time can use at most $f(n)$ bits of nondeterminism, so we need only generate all $2^{f(n)}$ such strings. Consider a modification of $\mathsf{CombinGenerate}(f(n))$ from above where the $f(n)$th bit has a glue that matches the seed of an input assembly, so that the input assemblies will grow at the end of each combinatorially assembled seed. By modifying the tree counters that raise the initial tile concentration, that we can modify the CRN-TAM decider from Theorem 5 to generate $cf(n)2^{f(n)}$ times as many tiles, so that the concentration of each tile is still $\Theta(V)$. Lastly, we modify the reject signal so that it is simply ε. So long as every string of nondeterministic choices is generated by $\mathsf{CombinGenerate}(f(n))$, the system will produce the accept signal if and only if the input x is in $\mathcal{L}$. We can perform a similar operation on the Turing machine that decides the complement of $\mathcal{L}$, except that we change its accept signal to the reject signal. Combining these two programs, we obtain a CRN-TAM decider for $\mathcal{L}$ that runs in $\Theta(f(n)+n+\log c)$ expected time, and succeeds whenever $\mathsf{CombinGenerate}(f(n))$ produces all $2^{f(n)}$ strings of length $f(n)$. □

More concretely, we find that there are languages that are decidable in polynomial time by CRN-TAM programs that are not decidable in polynomial time by Turing machines under standard complexity theoretic assumptions. Notice however that exponential *volume* might still be required; while we might evaluate all nondeterministic branches in parallel, each still needs space to operate.

Corollary 1. *Any language in* $\mathsf{NP} \cap \mathsf{coNP}$ *has a CRN-TAM decider that runs in* $\Theta(c\mathsf{poly}(n))$ *and succeeds with probability at least* $1 - e^{-c}$*.*

On its own, Theorem 7 demonstrates the computational power of the CRN-TAM. A careful reader will note that from the perspective of an actual laboratory experiment, the construction does not offer much beyond the capabilities of tile-only systems based on the aTAM. Although the theoretical formulation of the aTAM does not permit multiple assemblies, all experimental realizations thus far have many, many assemblies forming in the same reaction vessel [3,22]. Furthermore, the same tile set used for the combinatorial seed production step works just as well in the aTAM. From this point of view, the advantages offered by the CRN-TAM are nice, but not fundamentally different. Unlike the aTAM tile set, the CRN-TAM program can detect when an answer has been computed and exponentially amplify a signal indicating that. As we will see, the biggest advantage of the CRN-TAM is that is allows interaction between the assemblies while it is computing, allowing more powerful forms of parallel computation.

7 Towards Parallel Computation with Shared State and Open Questions

Even within the framework we have already described, a form of elementary shared state can be implemented with only a slight modification. Consider adding a special state tile with a distinguished removal signal ϕ and a reaction $\phi \to \boxed{\phi}$ that converts it into a tile that can bind to another state tile. If one assembly moves into the appropriate state, it can release ϕ and, by having $\boxed{\phi}$ attach onto another assembly, share some constant number of bits of information with another assembly. Furthermore, this signal can be amplified very quickly using a tree counter, allowing it to "turn the test tube red" and inform, with arbitrarily high probability, every other assembly that some assembly reached state ϕ. Rudimentary branch-and-bound, an algorithmic technique that has proved immensely useful for gaining dramatic speedups in optimization problems with only exponential time algorithms, can be implemented with this kind of rudimentary mechanism. In effect, we can use the CRN-TAM to simulate the types of parallel steps introduced by Lipton [16] for classical DNA computing that consisted of a series of laboratory operations on test tubes of DNA data, thus replacing a non-autonomous molecular computation by an autonomous one.

The parallel computational power of the CRN-TAM is far from limitless, however. Although we have been unable to show any concrete lower bounds, it seems very difficult to implement many general forms of parallel computation—such as languages decided by uniform circuits in time proportional to their depth, for example—because the well-mixed CRN is a difficult medium for passing information. In particular, sending more than constant-sized messages between different assemblies seems very difficult, since we must rely on chance to have the message arrive at its destination, and every other recipient must somehow recognize that the message is intended for another assembly.

Our work here has established a convenient kinetic model for the CRN-TAM and analyzed the time complexity of basic computational primitives, but many important questions remain open. Although we have shown the lower bound in Theorem 4, there is a gap between our lower bound (that copy-tolerant CRN-TAM deciders require $\Omega(t(n))$ volume) and our best construction, which requires $\mathcal{O}(t(n)s(n))$ space. The question of whether there exist copy-tolerant CRN-TAM deciders that require $o(t(n)s(n))$ volume remains open. Most questions related to efficient parallel computation in the CRN-TAM also remain open, and we have only considered the most basic ways of implementing it.

Acknowledgements. We acknowledge financial support from National Science Foundation grant CCF-1317694 and the Soli Deo Gloria Summer Undergraduate Research Fellowship at the California Institute of Technology. We also thank Dave Doty and Damien Woods for their insights.

References

1. Adleman, L., Cheng, Q., Goel, A., Huang, M.D.: Running time and program size for self-assembled squares. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, STOC 2001, pp. 740–748 (2001)
2. Aggarwal, G., Cheng, Q., Goldwasser, M.H., Kao, M.Y., de Espanes, P.M., Schweller, R.T.: Complexities for generalized models of self-assembly. SIAM J. Comput. **34**, 1493–1515 (2005)
3. Barish, R.D., Schulman, R., Rothemund, P.W.K., Winfree, E.: An information-bearing seed for nucleating algorithmic self-assembly. Proc. Natl. Acad. Sci. **106**, 6054–6059 (2009)
4. Bennett, C.H.: The thermodynamics of computation - a review. Int. J. Theoret. Phys. **21**, 905–940 (1982)
5. Cardelli, L.: Two-domain DNA strand displacement. Math. Struct. Comput. Sci. **23**, 247–271 (2013)
6. Cardelli, L., Zavattaro, G.: On the computational power of biochemistry. In: Horimoto, K., Regensburger, G., Rosenkranz, M., Yoshida, H. (eds.) AB 2008. LNCS, vol. 5147, pp. 65–80. Springer, Heidelberg (2008)
7. Chen, Y.J., Dalchau, N., Srinivas, N., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from DNA. Nat. Nanotechnol. **8**, 755–762 (2013)
8. Condon, A., Kirkpatrick, B., Maňuch, J.: Reachability bounds for chemical reaction networks and strand displacement systems. Nat. Comput. **13**, 499–516 (2014)
9. Doty, D.: Theory of algorithmic self-assembly. Commun. ACM **55**, 78–88 (2012)
10. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**, 2340–2361 (1977)
11. Gillespie, D.T.: Stochastic simulation of chemical kinetics. Annu. Rev. Phys. Chem. **58**, 35–55 (2007)

12. Ke, Y., Ong, L.L., Shih, W.M., Yin, P.: Three-dimensional structures self-assembled from DNA bricks. Science **338**, 1177–1183 (2012)
13. Kurtz, S., Mahaney, S., Royer, J., Simon, J.: Biological computing. In: Complexity Theory Retrospective II, pp. 179–195 (1997)
14. Lakin, M.R., Phillips, A.: Modelling, simulating and verifying turing-powerful strand displacement systems. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 130–144. Springer, Heidelberg (2011)
15. Lakin, M.R., Stefanovic, D., Phillips, A.: Modular verification of chemical reaction network encodings via serializability analysis. Theoret. Comput. Sci. **632**, 21–42 (2016)
16. Lipton, R.J.: DNA computations can have global memory. In: International Conference on Computer Design: VLSI in Computers and Processor, pp. 344–347 (1996)
17. Patitz, M.J.: An introduction to tile-based self-assembly and a survey of recent results. Nat. Comput. **13**, 195–224 (2013)
18. Phillips, A., Cardelli, L.: A programming language for composable DNA circuits. J. R. Soc. Interface **6**, S419–S436 (2009)
19. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
20. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science **332**, 1196–1201 (2011)
21. Qian, L., Winfree, E., Bruck, J.: Neural network computation with DNA strand displacement cascades. Nature **475**, 368–372 (2011)
22. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol. **2**, e424 (2004)
23. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, STOC 2000, pp. 459–468 (2000)
24. Schiefer, N., Winfree, E.: Universal computation and optimal construction in the chemical reaction network-controlled tile assembly model. In: Phillips, A., Yin, P. (eds.) DNA 2015. LNCS, vol. 9211, pp. 34–54. Springer, Heidelberg (2015)
25. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science **314**, 1585–1588 (2006)
26. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Nat. Comput. **7**, 615–633 (2008)
27. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. **107**, 5393–5398 (2010)
28. Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes. SIAM J. Comput. **36**, 1544–1569 (2007)
29. Wei, B., Dai, M., Yin, P.: Complex shapes self-assembled from single-stranded DNA tiles. Nature **485**, 623–626 (2012)
30. Winfree, E.: On the computational power of DNA annealing and ligation. In: DNA Computers. DIMACS Series in Discrete Mathematics and Computer Science, vol. 27, pp. 199–221. American Mathematical Society (1996)
31. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature **394**, 539–544 (1998)
32. Winfree, E., Yang, X., Seeman, N.C.: Universal computation via self-assembly of DNA: some theory and experiments. In: DNA Based Computers II. DIMACS Series in Discrete Mathematics and Computer Science, vol. 44, pp. 191–213. American Mathematical Society (1999)

33. Yin, P., Choi, H.M.T., Calvert, C.R., Pierce, N.A.: Programming biomolecular self-assembly pathways. Nature **451**, 318–322 (2008)
34. Zhang, D.Y., Hariadi, R.F., Choi, H.M.T., Winfree, E.: Integrating DNA strand-displacement circuitry with DNA tile self-assembly. Nat. Commun. **4**, Article no. 1965 (2013)
35. Zhang, D.Y., Turberfield, A.J., Yurke, B., Winfree, E.: Engineering entropy-driven reactions and networks catalyzed by DNA. Science **318**, 1121–1125 (2007)