Nat Comput (2016) 15:41–49
DOI 10.1007/s11047-015-9517-2

Producibility in hierarchical self-assembly

David Doty[1,2]

Published online: 20 August 2015

Abstract Three results are shown on producibility in the hierarchical model of tile self-assembly. It is shown that a simple greedy polynomial-time strategy decides whether an assembly α is producible. The algorithm can be optimized to use $O(|\alpha| \log^2 |\alpha|)$ time. Cannon et al. (STACS 2013: proceedings of the thirtieth international symposium on theoretical aspects of computer science. pp 172–184, 2013) showed that the problem of deciding if an assembly α is the unique producible terminal assembly of a tile system $\mathcal{T}$ can be solved in $O(|\alpha|^2|\mathcal{T}| + |\alpha||\mathcal{T}|^2)$ time for the special case of noncooperative "temperature 1" systems. It is shown that this can be improved to $O(|\alpha||\mathcal{T}| \log |\mathcal{T}|)$ time. Finally, it is shown that if two assemblies are producible, and if they can be overlapped consistently—i.e., if the positions that they share have the same tile type in each assembly—then their union is also producible.

Keywords Hierarchical · Self-assembly · Deterministic

1 Introduction

1.1 Background of the field

Winfree's abstract Tile Assembly Model (aTAM) (Winfree 1998) is a model of crystal growth through cooperative binding of square-like monomers called *tiles*, implemented experimentally (for the current time) by DNA (Winfree et al. 1998; Barish et al. 2009). In particular, it models the potentially algorithmic capabilities of tiles that can be designed to bind if and only if the total strength of attachment (summed over all binding sites, called *glues* on the tile) is at least a parameter τ, sometimes called the *temperature*. In particular, when the glue strengths are integers and $\tau = 2$, this implies that two strength 1 glues must cooperate to bind the tile to a growing assembly. Two assumptions are key: 1) growth starts from a single specially designated *seed* tile type, and 2) only individual tiles bind to an assembly, never larger assemblies consisting of more than one tile. We will refer to this model as the *seeded aTAM*. While violations of these assumptions are often viewed as errors in implementation of the seeded aTAM (Schulman and Winfree 2007, 2009), relaxing them results in a different model with its own programmable abilities. In the *hierarchical* [a.k.a. *multiple tile* (Aggarwal et al. 2004), *polyomino* (Winfree 2006; Luhrs 2010), *two-handed* (Cannon et al. 2013; Doty et al. 2010; Demaine et al. 2013)] *aTAM*, there is no seed tile, and an assembly is considered producible so long as two producible assemblies are able to attach to each other with strength at least τ, with all individual tiles being considered as "base case" producible assemblies. In either model, an assembly is considered *terminal* if nothing can attach to it; viewing self-assembly as a computation, terminal assembly(ies) are often interpreted to be the output. See Doty (2012) and Patitz (2012) for an introduction to recent work on these models.

The hierarchical aTAM has attracted considerable recent attention. It is coNP-complete to decide whether an assembly is the unique terminal assembly produced by a hierarchical tile system (Cannon et al. 2013). There are infinite shapes that can be assembled in the hierarchical aTAM but not the seeded aTAM, and vice versa, and there are finite shapes requiring strictly more tile types to assemble in the seeded aTAM than the hierarchical aTAM, and vice versa (Cannon et al. 2013). Despite this incomparability

✉ David Doty
doty@ucdavis.edu

1 California Institute of Technology, Pasadena, CA, USA

2 University of California, Davis, CA, USA

between the models for exact assembly of shapes, with a small blowup in scale, any seeded tile system can be simulated by a hierarchical tile system (Cannon et al. 2013), improving upon an earlier scheme that worked for restricted classes of seeded tile systems (Luhrs 2010). However, there is no single hierarchical aTAM tile set that can be used to simulate (at a larger scale factor) any other hierarchical aTAM system, i.e., it is not *intrinsically universal* (Demaine et al. 2013), unlike the seeded aTAM (Doty et al. 2012). It is possible to assemble an $n \times n$ square in a hierarchical tile system with $O(\log n)$ tile types that exhibits a very strong form of fault-tolerance in the face of spurious growth via strength 1 bonds (Doty et al. 2010). The parallelism of the hierarchical aTAM suggests the possibility that it can assemble shapes faster than the seeded aTAM, but it cannot for a wide class of tile systems (Chen and Doty 2012).

Interesting variants of the hierarchical aTAM introduce other assumptions to the model. The *multiple tile* model retains a seed tile and places a bound on the size of assemblies attaching to it (Aggarwal et al. 2004). Under this model, it is possible to modify a seeded tile system to be *self-healing*, that is, it correctly regrows when parts of itself are removed, even if the attaching assemblies that refill the removed gaps are grown without the seed (Winfree 2006). The model of *staged assembly* allows multiple test tubes to undergo independent growth, with excess incomplete assemblies washed away (e.g. purified based on size) and then mixed, with assemblies from each tube combining via hierarchical attachment (Demaine et al. 2008, 2013; Winslow 2013). The *RNase enzyme* model (Abel et al. 2010; Demaine et al. 2011; Patitz and Summers 2012) assumes some tile types to be made of RNA, which can be digested by an enzyme called RNase, leaving only the DNA tiles remaining, and possibly disconnecting what was previously a single RNA/DNA assembly into multiple DNA assemblies that can combine via hierarchical attachment. Introducing negative glue strengths into the hierarchical aTAM allows for "fuel-efficient" computation (Schweller and Sherman 2013). Allowing tiles with more complex geometry than squares enables hierarchical assembly to use significantly fewer tile types for assembly of $n \times n$ squares (Fu et al. 2012).

1.2 Contributions of this paper

We show three results on producibility in the hierarchical aTAM.

1. In the seeded aTAM, there is an obvious linear-time algorithm to test whether assembly α is producible by a tile system: starting from the seed, try to attach tiles until α is complete or no more attachments are possible. We show that in the hierarchical aTAM, a similar greedy strategy correctly identifies whether a given assembly is producible, though it is more involved to prove that it is correct. The idea is to start with all tiles in place as they appear in α, but with no bonds, and then to greedily bind attachable assemblies until α is assembled. It is not obvious that this works, since it is conceivable that assemblies must attach in a certain order for α to form, but the greedy strategy may pick another order and hit a dead-end in which no assemblies can attach. The algorithm can be optimized to use $O(|\alpha| \log^2 |\alpha|)$ time. This is shown in Sect. 3.
2. The temperature 1 Unique Production Verification (UPV) problem studied by Cannon et al. (2013) is the problem of determining whether assembly α is the unique producible terminal assembly of tile system $\mathcal{T}$, where $\mathcal{T}$ has temperature 1, meaning that all positive strength glues are sufficiently strong to attach any two assemblies. They give an algorithm that runs in $O(|\alpha|^2|\mathcal{T}| + |\alpha||\mathcal{T}|^2)$ time. Cannon et al. proved their result by using an $O(|\alpha|^2 + |\alpha||\mathcal{T}|)$ time algorithm for UPV that works in the seeded aTAM (Adleman et al. 2002), and then reduced the hierarchical temperature-1 UPV problem to $|\mathcal{T}|$ instances of the seeded UPV problem. We improve this result by showing that a faster $O(|\alpha| \log |\mathcal{T}|)$ time algorithm for the seeded UPV problem exists for the special case of temperature 1, and then we apply the technique of Cannon et al. relating the hierarchical problem to the seeded problem to improve the running time of the hierarchical algorithm to $O(|\alpha||\mathcal{T}| \log |\mathcal{T}|)$. This is shown in Sect. 4. Part of the conceptual significance of this algorithm lies in the details of the proof. In particular, we show a relationship between deterministic seeded assembly at temperature 1 and biconnected decomposition of the binding graph of an assembly using the Hopcroft–Tarjan algorithm (Hopcroft and Tarjan 1973). This relationship makes more precise the intuitive notion that determinism in temperature 1 systems with glue mismatches is enforced by "blocking." In particular, the tile that does the blocking must be a cut vertex of the binding graph and must be an ancestor of the blocked tile in the Hopcroft-Tarjan tree decomposition.
3. We show that if two assemblies α and β are producible in the hierarchical model, and if they can be overlapped consistently (i.e., if the positions that they share have the same tile type in each assembly), then their union $\alpha \cup \beta$ is producible. This is trivially true in the seeded model, but it requires more care to prove in the hierarchical model. It is conceivable *a priori* that although β is producible, β must assemble $\alpha \cap \beta$ in some order that is inconsistent with how α assembles $\alpha \cap \beta$. This is shown in Sect. 5.

1.3 Application of result of Sect. 5

The third result above (Theorem 5.1) has one application in showing a limitation on the power of hierarchical systems to assemble shapes "quickly."

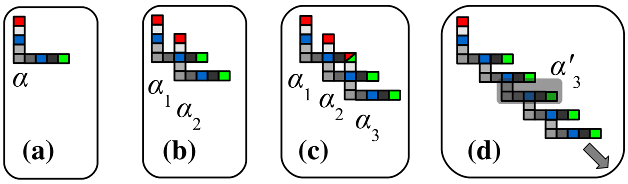

Fig. 1 **a** A producible assembly α. *Gray tiles* are all distinct types from each other, but *red*, *green*, and *blue* each represent one of three different tile types, so the *two blue tiles* are the same type. **b** By Theorem 5.1, $\alpha_1 \cup \alpha_2$ is producible, where $\alpha_1 = \alpha$ and $\alpha_2 = \alpha_1 + (2, -2)$, because they overlap in only one position, and they both have the *blue tile* type there. **c** α_1 and α_3 both have a tile at the same position, but the types are different (*red* in the case of α_1 and *green* in the case of α_3). **d** However, a subassembly α_i' of each new α_i can grow, enough to allow the translated equivalent subassembly α_{i+1}' of α_{i+1} to grow from α_i', so an infinite structure is producible. (Color figure online)

Theorem 5.1 shows that if assemblies α and β overlap consistently, then $\alpha \cup \beta$ is producible. What if $\alpha = \beta$? Suppose we have three copies of α, and label them each uniquely as $\alpha_1, \alpha_2, \alpha_3$. (See Fig. 1 for an example.) Suppose further that α_2 overlaps consistently with α_1 when translated by some non-zero vector $\vec{v}$. Then we know that $\alpha_1 \cup \alpha_2$ is producible. Suppose that α_3 is α_2 translated by $\vec{v}$, or equivalently it is α_1 translated by $2\vec{v}$. Then $\alpha_2 \cup \alpha_3$ is producible, since this is merely a translated copy of $\alpha_1 \cup \alpha_2$. It seems intuitively that $\alpha_1 \cup \alpha_2 \cup \alpha_3$ should be producible as well. However, while α_1 overlaps consistently with α_2, and α_2 overlaps consistently with α_3, it could be the case that α_3 intersects α_1 inconsistently, i.e., they share a position but put a different tile type at that position. In this case $\alpha_1 \cup \alpha_2 \cup \alpha_3$ is undefined.

In the example of Fig. 1, although $\alpha_1 \cup \alpha_2 \cup \alpha_3$ is not producible (in fact, not even defined), "enough" of α_3 (say, $\alpha_3' \sqsubset \alpha_3$) can grow off of $\alpha_1 \cup \alpha_2$ to allow a fourth copy α_4' to begin to grow to an assembly to which a fifth copy α_5' can attach, etc., so that an infinite assembly can grow by "pumping" additional copies of α_3'. Is this always possible? In other words, is it the case that if α is a producible assembly of a hierarchical TAS $\mathcal{T}$, and α overlaps consistently with some non-zero translation of itself, then $\mathcal{T}$ necessarily produces arbitrarily large assemblies?

This question was answered affirmatively by Chen et al. (2015). This in turn settled an open question posed by Chen and Doty (2012), who showed that as long as a hierarchical TAS does not produce assemblies that consistently overlap any translation of themselves, then the TAS cannot uniquely produce any shape in time sublinear in its diameter.

2 Formal definition of the abstract tile assembly model

This section gives a terse definition of the abstract Tile Assembly Model (aTAM, Winfree (1998)). This is not a tutorial; for readers unfamiliar with the aTAM, (Rothemund and Winfree 2000) gives an excellent introduction to the model.

Fix an alphabet Σ. Σ^* is the set of finite strings over Σ. Given a discrete object O, $\langle O \rangle$ denotes a standard encoding of O as an element of Σ^*. $\mathbb{Z}$, $\mathbb{Z}^+$, and $\mathbb{N}$ denote the set of integers, positive integers, and nonnegative integers, respectively. For a set A, $\mathcal{P}(A)$ denotes the power set of A. Given $A \subseteq \mathbb{Z}^2$, the *full grid graph* of A is the undirected graph $G_A^{\mathrm{f}} = (V, E)$, where $V = A$, and for all $u, v \in V$, $\{u, v\} \in E \iff \|u - v\|_2 = 1$; i.e., if and only if u and v are adjacent on the integer Cartesian plane. A *shape* is a set $S \subseteq \mathbb{Z}^2$ such that G_S^{f} is connected.

A *tile type* is a tuple $t \in (\Sigma^* \times \mathbb{N})^4$; i.e., a unit square with four sides listed in some standardized order, each side having a *glue label* (a.k.a. *glue*) $\ell \in \Sigma^*$ and a nonnegative integer *strength*, denoted $str(\ell)$. For a set of tile types T, let $\Lambda(T) \subset \Sigma^*$ denote the set of all glue labels of tile types in T. If a glue has strength 0, we say it is *null*, and if a positive-strength glue facing some direction does not appear on some tile type in the opposite direction, we say it is *functionally null*. We assume that all tile sets in this paper contain no functionally null glues.[1] Let $\{\mathsf{N}, \mathsf{S}, \mathsf{E}, \mathsf{W}\}$ denote the *directions* consisting of unit vectors $\{(0,1), (0,-1), (1,0), (-1,0)\}$. Given a tile type t and a direction $d \in \{\mathsf{N}, \mathsf{S}, \mathsf{E}, \mathsf{W}\}$, $t(d) \in \Lambda(T)$ denotes the glue label on t in direction d. We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile*. An *assembly* is a nonempty connected arrangement of tiles on the integer lattice $\mathbb{Z}^2$, i.e., a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$ such that $G^{\mathrm{f}}_{\mathrm{dom}\ \alpha}$ is connected and $\mathrm{dom}\ \alpha \neq \emptyset$. The *shape of* α is $\mathrm{dom}\ \alpha$. Write $|\alpha|$ to denote $|\mathrm{dom}\ \alpha|$. Given two assemblies $\alpha, \beta : \mathbb{Z}^2 \dashrightarrow T$, we say α is a *subassembly* of β, and we write $\alpha \sqsubseteq \beta$, if $\mathrm{dom}\ \alpha \subseteq \mathrm{dom}\ \beta$ and, for all points $p \in \mathrm{dom}\ \alpha$, $\alpha(p) = \beta(p)$.

Given two assemblies α and β, we say α and β are *equivalent up to translation*, written $\alpha \simeq \beta$, if there is a vector $\vec{x} \in \mathbb{Z}^2$ such that $\mathrm{dom}\ \alpha = \mathrm{dom}\ \beta + \vec{x}$ (where for $A \subseteq \mathbb{Z}^2$, $A + \vec{x}$ is defined to be $\{p + \vec{x} \mid p \in A\}$) and for all $p \in \mathrm{dom}\ \beta$, $\alpha(p + \vec{x}) = \beta(p)$. In this case we say that β is a *translation* of α. We have fixed assemblies at certain positions on $\mathbb{Z}^2$ only for mathematical convenience in some contexts, but of course real assemblies float freely in solution and do not have a fixed position.

Let α be an assembly and let $p \in \mathrm{dom}\ \alpha$ and $d \in \{\mathsf{N}, \mathsf{S}, \mathsf{E}, \mathsf{W}\}$ such that $p + d \in \mathrm{dom}\ \alpha$. Let $t = \alpha(p)$ and

[1] This assumption does not affect the results of this paper. It is irrelevant for Theorem 5.1 or the correctness of the algorithms in the other theorems. It also does not affect the running time results for algorithms taking a TAS as input, because we can preprocess T in linear time to find and set to null any functionally null glues. The number of glues is $O(|T|)$, and we assume that each glue from glue set G is an integer in the set $\{0, \ldots, |G| - 1\}$. We can use a Boolean array of size $|G|$ to determine in time $O(|T|)$ which glues appear on the north that do not appear on the south of some tile type. Repeat this for each of the remaining three directions. Then replace all functionally null glues in T with null glues, which takes time $O(|T|)$. To do this replacement in an assembly α takes time $O(|\alpha|)$.

$t' = \alpha(p + d)$. We say that the tiles t and t' at positions p and $p + d$ *interact* if $t(d) = t'(-d)$ and $str(t(d)) > 0$, i.e., if the glue labels on their abutting sides are equal and have positive strength. Each assembly α induces a *binding graph* G^{b}_{α}, a grid graph $G = (V_\alpha, E_\alpha)$, where $V_\alpha = \mathrm{dom}\ \alpha$, and $\{p_1, p_2\} \in E_\alpha \iff \alpha(p_1)$ interacts with $\alpha(p_2)$.[2] Given $\tau \in \mathbb{Z}^+$, α is τ-*stable* if every cut of G^{b}_{α} has weight at least τ, where the weight of an edge is the strength of the glue it represents. That is, α is τ-stable if at least energy τ is required to separate α into two parts. When τ is clear from context, we say α is *stable*.

2.1 Seeded aTAM

A *seeded tile assembly system* (seeded TAS) is a triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is the finite, τ-stable *seed assembly*, and $\tau \in \mathbb{Z}^+$ is the *temperature*. Let $|\mathcal{T}|$ denote $|T|$. If $\mathcal{T}$ has a single seed tile $s \in T$ (i.e., $\sigma(0,0) = s$ for some $s \in T$ and is undefined elsewhere), then we write $\mathcal{T} = (T, s, \tau)$. Given two τ-stable assemblies $\alpha, \beta : \mathbb{Z}^2 \dashrightarrow T$, we write $\alpha \to_1^{\mathcal{T}} \beta$ if $\alpha \sqsubseteq \beta$ and $|\mathrm{dom}\ \beta \setminus \mathrm{dom}\ \alpha| = 1$. In this case we say α $\mathcal{T}$*-produces* β *in one step*.[3] If $\alpha \to_1^{\mathcal{T}} \beta$, $\mathrm{dom}\ \beta \setminus \mathrm{dom}\ \alpha = \{p\}$, and $t = \beta(p)$, we write $\beta = \alpha + (p \mapsto t)$.

A sequence of $k \in \mathbb{Z}^+$ assemblies $\vec{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{k-1})$ is a $\mathcal{T}$*-assembly sequence* if, for all $1 \leq i < k$, $\alpha_{i-1} \to_1^{\mathcal{T}} \alpha_i$. We write $\alpha \to^{\mathcal{T}} \beta$, and we say α $\mathcal{T}$*-produces* β (in 0 or more steps) if there is a $\mathcal{T}$-assembly sequence $\vec{\alpha} = (\alpha, \alpha_1, \alpha_2, \ldots, \alpha_{k-1} = \beta)$ of length $k = |\mathrm{dom}\ \beta \setminus \mathrm{dom}\ \alpha| + 1$. We say α is $\mathcal{T}$*-producible* if $\sigma \to^{\mathcal{T}} \alpha$, and we write $\mathcal{A}[\mathcal{T}]$ to denote the set of $\mathcal{T}$-producible assemblies. The relation $\to^{\mathcal{T}}$ is a partial order on $\mathcal{A}[\mathcal{T}]$ (Rothemund 2001; Lathrop et al. 2009).

An assembly α is $\mathcal{T}$*-terminal* if α is τ-stable and $\partial^{\mathcal{T}}\alpha = \emptyset$. We write $\mathcal{A}_\Box[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ to denote the set of $\mathcal{T}$-producible, $\mathcal{T}$-terminal assemblies.

A seeded TAS $\mathcal{T}$ is *directed (a.k.a., deterministic, confluent)* if the poset $(\mathcal{A}[\mathcal{T}], \to^{\mathcal{T}})$ is directed; i.e., if for each $\alpha, \beta \in \mathcal{A}[\mathcal{T}]$, there exists $\gamma \in \mathcal{A}[\mathcal{T}]$ such that $\alpha \to^{\mathcal{T}} \gamma$ and $\beta \to^{\mathcal{T}} \gamma$.[4] We say that $\mathcal{T}$ *uniquely produces* α if $\mathcal{A}_\Box[\mathcal{T}] = \{\alpha\}$.

2.2 Hierarchical aTAM

A *hierarchical tile assembly system* (hierarchical TAS) is a pair $\mathcal{T} = (T, \tau)$, where T is a finite set of tile types, and $\tau \in \mathbb{Z}^+$ is the *temperature*. Let $\alpha, \beta : \mathbb{Z}^2 \dashrightarrow T$ be two assemblies. Say that α and β are *nonoverlapping* if $\mathrm{dom}\ \alpha \cap \mathrm{dom}\ \beta = \emptyset$. If α and β are nonoverlapping assemblies, define $\alpha \cup \beta$ to be the assembly γ defined by $\gamma(p) = \alpha(p)$ for all $p \in \mathrm{dom}\ \alpha$, $\gamma(p) = \beta(p)$ for all $p \in \mathrm{dom}\ \beta$, and $\gamma(p)$ is undefined for all $p \in \mathbb{Z}^2 \setminus (\mathrm{dom}\ \alpha \cup \mathrm{dom}\ \beta)$. An assembly γ is *singular* if $\gamma(p) = t$ for some $p \in \mathbb{Z}^2$ and some $t \in T$ and $\gamma(p')$ is undefined for all $p' \in \mathbb{Z}^2 \setminus \{p\}$. Given a hierarchical TAS $\mathcal{T} = (T, \tau)$, an assembly γ is $\mathcal{T}$*-producible* if either (1) γ is singular, or (2) there exist producible nonoverlapping assemblies α and β such that $\gamma = \alpha \cup \beta$ and γ is τ-stable. In the latter case, write $\alpha + \beta \to_1^{\mathcal{T}} \gamma$. An assembly α is $\mathcal{T}$*-terminal* if for every producible assembly β such that α and β are nonoverlapping, $\alpha \cup \beta$ is not τ-stable.[5] Define $\mathcal{A}[\mathcal{T}]$ to be the set of all $\mathcal{T}$-producible assemblies. Define $\mathcal{A}_\Box[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ to be the set of all $\mathcal{T}$-producible, $\mathcal{T}$-terminal assemblies. A hierarchical TAS $\mathcal{T}$ is *directed (a.k.a., deterministic, confluent)* if $|\mathcal{A}_\Box[\mathcal{T}]| = 1$. We say that $\mathcal{T}$ *uniquely produces* α if $\mathcal{A}_\Box[\mathcal{T}] = \{\alpha\}$.

Let $\mathcal{T}$ be a hierarchical TAS, and let $\widehat{\alpha} \in \mathcal{A}[\mathcal{T}]$ be a $\mathcal{T}$-producible assembly. An *assembly tree* Υ of $\widehat{\alpha}$ is a full binary tree with $|\widehat{\alpha}|$ leaves, whose nodes are labeled by $\mathcal{T}$-producible assemblies, with $\widehat{\alpha}$ labeling the root, singular assemblies labeling the leaves, and node u labeled with γ having children u_1 labeled with α and u_2 labeled with β, with the requirement that $\alpha + \beta \to_1^{\mathcal{T}} \gamma$. That is, Υ represents one possible pathway through which $\widehat{\alpha}$ could be produced from individual tile types in $\mathcal{T}$. Let $\Upsilon(\mathcal{T})$ denote the set of all assembly trees of $\mathcal{T}$. If α is a descendant node of β in an assembly tree of $\mathcal{T}$, write $\alpha \to^{\mathcal{T}} \beta$. Say that an assembly tree is $\mathcal{T}$*-terminal* if its root is a $\mathcal{T}$-terminal assembly. Let $\Upsilon_\Box(\mathcal{T})$ denote the set of all $\mathcal{T}$-terminal assembly trees of $\mathcal{T}$. Note that even a directed hierarchical TAS can have multiple terminal assembly trees that all have the same root terminal assembly.

When $\mathcal{T}$ is clear from context, we may omit $\mathcal{T}$ from the notation above and instead write $\to_1$, $\to$, $\partial\alpha$, *assembly sequence*, *produces*, *producible*, and *terminal*.

3 Efficient verification of production

Let S be a finite set, and let $\mathcal{P}(S)$ be its power set. A *partition* of S is a collection $\mathcal{C} = \{C_1, \ldots, C_k\} \subseteq \mathcal{P}(S)$ such that $\bigcup_{i=1}^{k} C_i = S$ and for all $i \neq j$, $C_i \cap C_j = \emptyset$. A

[2] For $G^{\mathrm{f}}_{\mathrm{dom}\ \alpha} = (V_{\mathrm{dom}\ \alpha}, E_{\mathrm{dom}\ \alpha})$ and $G^{\mathrm{b}}_{\alpha} = (V_\alpha, E_\alpha)$, G^{b}_{α} is a spanning subgraph of $G^{\mathrm{f}}_{\mathrm{dom}\ \alpha}$: $V_\alpha = V_{\mathrm{dom}\ \alpha}$ and $E_\alpha \subseteq E_{\mathrm{dom}\ \alpha}$.

[3] Intuitively $\alpha \to_1^{\mathcal{T}} \beta$ means that α can grow into β by the addition of a single tile; the fact that we require both α and β to be τ-stable implies in particular that the new tile is able to bind to α with strength at least τ. It is easy to check that had we instead required only α to be τ-stable, and required that the cut of β separating α from the new tile has strength at least τ, then this implies that β is also τ-stable.

[4] The following two convenient characterizations of "directed" are routine to verify. $\mathcal{T}$ is directed if and only if $|\mathcal{A}_\Box[\mathcal{T}]| = 1$. $\mathcal{T}$ is *not* directed if and only if there exist $\alpha, \beta \in \mathcal{A}[\mathcal{T}]$ and $p \in \mathrm{dom}\ \alpha \cap \mathrm{dom}\ \beta$ such that $\alpha(p) \neq \beta(p)$.

[5] The restriction on overlap is a model of a chemical phenomenon known as *steric hindrance* (Wade 1991, Section 5.11) or, particularly when employed as a design tool for intentional prevention of unwanted binding in synthesized molecules, *steric protection* (Heller and Pugh 1954, 1960; Goto et al. 2000).

Algorithm 1 IS-PRODUCIBLE-ASSEMBLY(α, τ)

1: **input:** assembly α and temperature τ
2: $\mathcal{C} \leftarrow \{ \{v\} \mid v \in \mathrm{dom}\, \alpha \}$ // *(positions defining) subassemblies of* α
3: **while** $|\mathcal{C}| > 1$ **do**
4: **if** there exist $C_i, C_j \in \mathcal{C}$ with glues between C_i and C_j of total strength at least τ **then**
5: $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{C_i, C_j\}) \cup \{C_i \cup C_j\}$
6: **else**
7: **print** "α is not producible" and **exit**
8: **end if**
9: **end while**
10: **print** "α is producible"

hierarchical division of S is a full binary tree Υ (a tree in which every internal node has exactly two children) whose nodes represent subsets of S, such that the root of Υ represents S, the $|S|$ leaves of Υ represent the singleton sets $\{x\}$ for each $x \in S$, and each internal node has the property that its set is the (disjoint) union of its two childrens' sets.

Lemma 3.1 *Let S be a finite set with $|S| \geq 2$. Let Υ be any hierarchical division of S*, and let *$\mathcal{C}$ be any partition of S other than $\{S\}$. Then there exist $C_1, C_2 \in \mathcal{C}$ with $C_1 \neq C_2$, and there exist $C_1' \subseteq C_1$ and $C_2' \subseteq C_2$, such that C_1' and C_2' are siblings in Υ.*

Proof First, label each leaf $\{x\}$ of Υ with the unique element $C_i \in \mathcal{C}$ such that $x \in C_i$. Next, iteratively label internal nodes according to the following rule: while there exist two children of a node u that have the same label, assign that label to u. Notice that this rule preserves the invariant that each labeled node u (representing a subset of S) is a subset of the set its label represents. Continue until no node has two identically-labeled children. $\mathcal{C}$ contains only proper subsets of S, so the root (which is the set S) cannot be contained in any of them, implying the root will remain unlabeled. Follow any path starting at the root, always following an unlabeled child, until both children of the current internal node are labeled. (The path may vacuously end at the root.) Such a node is well-defined since at least all leaves are labeled. By the stopping condition stated previously, these children must be labeled differently. The children are the witnesses C_1' and C_2', with their labels having the values C_1 and C_2, testifying to the truth of the lemma. □

Lemma 3.1 will be useful when we view Υ as an assembly tree for some producible assembly α, and we view $\mathcal{C}$ as a partially completed attempt to construct another assembly tree for α, where each element of $\mathcal{C}$ is a subassembly that has been produced so far.

When we say "by monotonicity", this refers to the fact that glue strengths are nonnegative, which implies that if two assemblies α and β can attach, the addition of more tiles to either α or β cannot *prevent* this binding, so long as the additional tiles do not overlap the other assembly.

We want to solve the following problem: given an assembly α and temperature τ, is α producible in the hierarchical aTAM at temperature τ?[6] The algorithm IS-PRODUCIBLE-ASSEMBLY (Algorithm 1) solves this problem.

Theorem 3.2 *There is an $O(|\alpha| \log^2 |\alpha|)$ time algorithm deciding whether an assembly α is producible at temperature τ in the hierarchical aTAM.*

Proof **Correctness:** IS-PRODUCIBLE-ASSEMBLY works by building up the initially edge-free graph with the tiles of α as its nodes (the algorithm stores the nodes as points in $\mathbb{Z}^2$, but α would be used in step 4 to get the glues and strengths between tiles at adjacent positions), stopping when the graph becomes connected. The order in which connected components (implicitly representing assemblies) are removed from and added to $\mathcal{C}$ implicitly defines a particular assembly tree with α at the root (for every C_1, C_2 processed in line 5, the assembly $\alpha \upharpoonright (C_1 \cup C_2)$ is a parent of $\alpha \upharpoonright C_1$ and $\alpha \upharpoonright C_2$ in the assembly tree). Therefore, if the algorithm reports that α is producible, then it is. Conversely, suppose that α is producible via assembly tree Υ. Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be the set of assemblies at some iteration of the loop at line 3. It suffices to show that some pair of assemblies C_i and C_j are connected by glues with strength at least τ. By Lemma 3.1, there exist C_i and C_j with subsets $C_i' \subseteq C_i$ and $C_j' \subseteq C_j$ such that C_i' and C_j' are sibling nodes in Υ. Because they are siblings, the glues between C_i' and C_j' have strength at least τ. By monotonicity these glues suffice to bind C_i to C_j, so IS-PRODUCIBLE-ASSEMBLY is correct.

Running time: Let $n = |\alpha|$. The running time of the IS-PRODUCIBLE-ASSEMBLY (Algorithm 1) is polynomial in n, but the algorithm can be optimized to improve the running time to $O(n \log^2 n)$ by careful choice of data structures. IS-PRODUCIBLE-ASSEMBLY-FAST (Algorithm 2) shows pseudocode for this optimized implementation, which we now describe. Let $n = |\alpha|$. Instead of searching over all pairs of assemblies, only search adjacent pairs. There are $O(n)$ such pairs since a grid graph has degree at most 4 (hence $O(n)$ edges), and the number of edges in the full grid graph of α is an upper bound on the number of adjacent assemblies at any time. This can be encoded in a dynamically changing graph G_c whose nodes are the current set of assemblies and whose edges connect those assemblies that are adjacent.

Each edge of G_c stores the total glue strength between the assemblies. Whenever two assemblies C_1 and C_2, with $|C_1| \geq |C_2|$ without loss of generality, are combined to form a new assembly, G_c is updated by removing C_2, merging its edges with those of C_1, and for any edges they already share (i.e., the neighbor on the other end of the edge is the

[6] We do not need to give the tile set T as input because the tiles in α implicitly define a tile set, and the presence of extra tile types in T that do not appear in α cannot affect its producibility.

same), summing the strengths on the edges. Each update of an edge (adding it to C_1, or finding it in C_1 to update its strength) can be done in $O(\log n)$ time using a tree set data structure to store neighbors for each assembly.

We claim that the total number of such updates of all edges is $O(n \log n)$ over all time, or amortized $O(\log n)$ updates per iteration of the outer loop. To see why, observe that the number of edges an assembly has is at most linear in its size, so the number of new edges that must be added to C_1, or existing edges in C_1 whose strengths must be updated, is at most (within a constant) the size of the smaller component C_2. The total number of edge updates is then, if Υ is the assembly tree discovered by the algorithm, $\sum_{\text{nodes} u \in \Upsilon} \min\{|\text{left}(u)|, |\text{right}(u)|\}$, where $|\text{left}(u)|$ and $|\text{right}(u)|$ respectively refer to the number of leaves of u's left and right subtrees. For a given number n of leaves, this sum is maximized with a balanced tree, and in that case (summing over all levels of the tree) is $\sum_{i=0}^{\log n} 2^i (n/2^i) = O(n \log n)$. So the total time to update all edges is $O(n \log^2 n)$.

As for actually finding C_1 and C_2, each iteration of the outer loop, we can look at *any* pair of adjacent assemblies with sufficient connection strength. So in addition to storing the edges in a tree-backed set data structure, store them also in one of two linked lists: H and L in the algorithm, for "high" (strength $\geq \tau$) and "low" (strength $< \tau$), with each edge storing a pointer to its node in the linked list for $O(1)$ time removal (and also to its node in the tree-backed set for $O(\log n)$ time removal). We can simply choose an arbitrary edge from H to be the next pair of connected components to attach. We update the keys containing C_1 whose connection strength changed and removing those containing C_2 but not C_1. The edges whose connection strength changed correspond to precisely those neighbors that C_1 and C_2 shared before being merged. Therefore $|C_2|$ is an upper bound on the number of edge updates required. Thus the amortized number of linked list updates is $O(\log n)$ per iteration of the outer loop by the same argument as above. Since we can have each edge $\{C_1, C_2\}$ store a pointer to its node in the linked list to which it belongs, each list update can be done in $O(1)$ time. Thus each iteration takes amortized time $O(\log n)$.

The algorithm Is-Producible-Assembly-Fast (Algorithm 2) implements this optimized idea. The terminology for data structure operations is taken from Thomas (2001). Note that the way we remove C_1 and C_2 and add their union is to simply delete C_2 and then update C_1 to contain C_2's edges. The graph G_c discussed above is $G_c = (V_c, E_c)$ where V_c and E_c are variables in Is-Producible-Assembly-Fast.

Summarizing the analysis, each data structure operation takes time $O(\log n)$ with appropriate choice of a backing data structure. The two outer loops (lines 6 and 10) take $O(n)$ iterations. The inner loop (line 17) runs for amortized $O(\log n)$ iterations, and its body executes a constant number of $O(\log n)$ and $O(1)$ time operations. Therefore the total running time is $O(n \log^2 n)$. □

4 Efficient verification of temperature 1 unique production

This section shows that there is an algorithm, faster than the previous known algorithm (Cannon et al. 2013), that solves the *temperature 1 unique producibility verification* (UPV_1) problem: given an assembly α and a temperature-1 hierarchical tile system $\mathcal{T}$, decide if α is the unique producible, terminal assembly of $\mathcal{T}$. This is done by showing an algorithm for the UPV_1 problem in the seeded model (which is faster than the general-temperature algorithm of Adleman et al. 2002), and then applying the technique of Cannon et al. (2013) relating producibility and terminality in the temperature 1 seeded and hierarchical models.

Let the decision problems sUPV_1 and hUPV_1 be represented by the language $\{(\mathcal{T}, \alpha) | \mathcal{A}_\Box[\mathcal{T}] = \{\alpha\}\}$, where $\mathcal{T}$ is a temperature 1 seeded TAS in the former case and a temperature 1 hierarchical TAS in the latter case. To simplify the time analysis we assume $|\mathcal{T}| = O(|\alpha|)$.

The following is the only result in this paper on the seeded aTAM.

Theorem 4.1 *There is an algorithm that solves the* sUPV_1 *problem in time* $O(|\alpha| \log |\mathcal{T}|)$.

Proof Let $\mathcal{T} = (T, s, 1)$ and α be a instance of the sUPV_1 problem. We first check that every tile in α appears in T, which can be done in time $O(|\alpha| \log |T|)$ by storing elements of T in a data structure supporting $O(\log n)$ time access. In the seeded aTAM at temperature 1, α is producible if and only if it contains the seed s and its binding graph is connected, which can be checked in time $O(|\alpha|)$. We must also verify that α is terminal, which is true if and only if all glues on unbound sides are null, checkable in time $O(|\alpha|)$.

Once we have verified that α is producible and terminal, it remains to verify that $\mathcal{T}$ uniquely produces α. Adleman et al. (2002) showed that this is true (at any temperature) if and only if, for every position $p \in \text{dom}\, \alpha$, if $\alpha_p \sqsubset \alpha$ is the maximal producible subassembly of α such that $p \notin \text{dom}\, \alpha_p$, then $\alpha(p)$ is the only tile type attachable to α_p at position p. They solve the problem by producing each such α_p and checking whether there is more than one tile type attachable to α_p at p. We use a similar approach, but we avoid the cost of producing each α_p by exploiting special properties of temperature 1 producibility.

Given $p, q \in \text{dom}\, \alpha$ such that $p \neq q$, write $p \prec q$ if, for every producible assembly β, $q \in \text{dom}\, \beta \Rightarrow p \in \text{dom}\, \beta$, i.e., the tile at position p must be present before the tile at position q can be attached. We must check each $p \in \text{dom}\, \alpha$ and each position $q \in \text{dom}\, \alpha$ adjacent to p such that $p \not\prec q$

Algorithm 2 IS-PRODUCIBLE-ASSEMBLY-FAST(α, τ)

input: assembly α and temperature τ
$V_c \leftarrow \{ \{v\} \mid v \in \text{dom } \alpha \}$ // *(positions defining) subassemblies of* α
$E_c \leftarrow \{\{\{u\},\{v\}\} \mid \{u\} \in V_c$ and $\{v\} \in V_c$ and u and v are adjacent and interact$\}$
$H \leftarrow$ empty linked list // *pairs of subassemblies binding with strength* $\geq \tau$
$L \leftarrow$ empty linked list // *pairs of subassemblies binding with strength* $< \tau$
for all $\{\{u\},\{v\}\} \in E_c$ **do**
  $w(\{u\},\{v\}) \leftarrow$ strength of glue binding $\alpha(u)$ and $\alpha(v)$
  append $\{\{u\},\{v\}\}$ to L if $w(\{u\},\{v\}) < \tau$, and append to H otherwise
end for
while $|V_c| > 1$ **do**
  if H is empty **then**
    print "α is not producible" and **exit**
  end if
  $\{C_1, C_2\} \leftarrow$ first element of H // *assume* $|C_1| \geq |C_2|$ *without loss of generality*
  remove $\{C_1, C_2\}$ from H
  remove C_2 from V_c
  for all neighbors C of C_2 **do**
    remove $\{C_2, C\}$ from E_c and H or L
    if $\{C_1, C\} \in E_c$ **then**
      $w(C_1, C) \leftarrow w(C_1, C) + w(C_2, C)$
      if $w(C_1, C) \geq \tau$ and $\{C_1, C\} \in L$ **then**
        remove $\{C_1, C\}$ from L and add it to H
      end if
    else
      $w(C_1, C) \leftarrow w(C_2, C)$
      add $\{C_1, C\}$ to E_c and to H if $w(C_1, C) \geq \tau$ and to L otherwise
    end if
  end for
end while
print "α is producible"

to see whether a tile type $t \neq \alpha(p)$ shares a positive-strength glue with $\alpha(q)$ in direction $q - p$ (i.e., whether, if $\alpha(p)$ were not present, t could attach at p instead). If we know which positions q adjacent to p satisfy $p \not\prec q$, this check can be done in time $O(\log |T|)$ with appropriate choice of data structure, implying total time $O(|\alpha| \log |T|)$ over all positions $p \in \text{dom } \alpha$. It remains to show how to determine which adjacent positions $p, q \in \text{dom } \alpha$ satisfy $p \prec q$.

Recall that a *cut vertex* of a connected graph is a vertex whose removal disconnects the graph, and a subgraph is *biconnected* if the removal of any single vertex from the subgraph leaves it connected. Every graph can be decomposed into a tree of biconnected components, with cut vertices connecting different biconnected components (and belonging to all biconnected components that they connect). If p is not a cut vertex of the binding graph of α, then dom α_p is simply dom $\alpha \setminus \{p\}$ (i.e., it is possible to produce the entire assembly α except for position p) because, for all $q \in \text{dom } \alpha \setminus \{p\}$, $p \not\prec q$. If p is a cut vertex, then $p \prec q$ if and only if removing p from the binding graph of α places q and the seed position in two different connected components, since the connected component containing the seed after removing p corresponds precisely to α_p.

Run the linear time Hopcroft–Tarjan algorithm (Hopcroft and Tarjan 1973) for decomposing the binding graph of α into a tree of its biconnected components, which also identifies which vertices in the graph are cut vertices and which biconnected components they connect. Recall that the Hopcroft-Tarjan algorithm is an augmented depth-first search. Root the tree with s's biconnected component (i.e., start the depth-first search there), so that each component has a parent component and child components. In particular, each cut vertex p has a "parent" biconnected component and $k \geq 1$ "child" biconnected components. Removing p will separate the graph into $k + 1$ connected components: the k subtrees and the remaining nodes connected to the parent biconnected component of p. Thus $p \prec q$ if and only if p is a cut vertex and q is contained in the subtree rooted at p.

This check can be done for all positions p and their ≤ 4 adjacent positions q in linear time by "weaving" the checks into the Hopcroft-Tarjan algorithm. As the depth-first search executes, each vertex p is marked as either *unvisited*, *visiting* (meaning the search is currently in a subtree rooted at p), or *visited* (meaning the search has visited and exited the subtree rooted at p). If p is marked as visited or unvisited when q is processed, then q is not in the subtree under p. If p is marked as visiting when q is processed, then q is in p's subtree.

At the time q is visited during the Hopcroft-Tarjan algorithm, it may not yet be known whether p is a cut vertex. To account for this, simply run the Hopcroft-Tarjan algorithm first to label all cut vertices, then run a second depth-first search (visiting the nodes in the same order as the first depth-first search), doing the checks described previously and using the cut vertex information obtained from the Hopcroft-Tarjan algorithm. □

Theorem 4.2 *There is an algorithm that solves the* $\textsf{hUPV}_1$ *problem in time* $O(|\alpha||\mathcal{T}| \log |\mathcal{T}|)$.

Proof Cannon et al. (2013) showed that a temperature 1 hierarchical TAS $\mathcal{T} = (T, 1)$ uniquely produces α if and only if, for each $s \in T$, the seeded TAS $\mathcal{T}_s = (T, s, 1)$ uniquely produces α. Therefore, the $\textsf{hUPV}_1$ problem can be solved by calling the algorithm of Theorem 4.1 $|\mathcal{T}|$ times, resulting in a running time of $O(|\alpha||\mathcal{T}| \log |\mathcal{T}|)$. □

5 Consistent unions of producible assemblies are producible

Throughout this section, fix a hierarchical TAS $\mathcal{T} = (T, \tau)$. Let α, β be assemblies. We say α and β are *consistent* if $\alpha(p) = \beta(p)$ for all points $p \in \text{dom } \alpha \cap \text{dom } \beta$. If α and β are consistent, let $\alpha \cup \beta$ be defined as the assembly $(\alpha \cup \beta)(p) = \alpha(p)$ if α is defined, and $(\alpha \cup \beta)(p) = \beta(p)$ if $\alpha(p)$

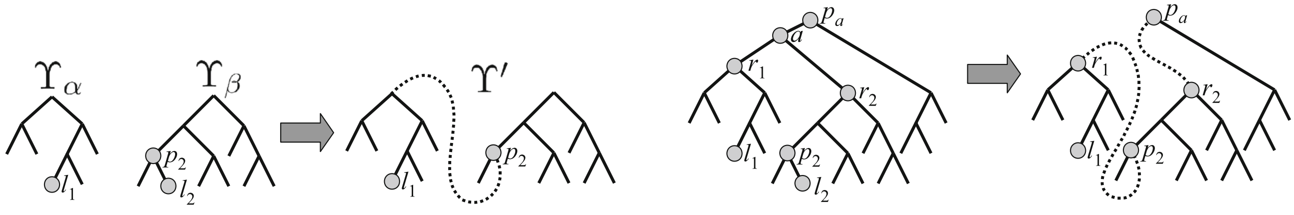

Fig. 2 Constructing assembly tree for $\alpha \cup \beta$ from assembly trees for α and β. **a** First operation to combine the assembly trees for α and β. l_1 and l_2 are two leaves representing the same position in dom $\alpha \cap$ dom β. **b** Operation to eliminate one of two leaves l_1 and l_2 representing the same tile in the tree while preserving that all attachments are stable

is undefined. If α and β are not consistent, let $\alpha \cup \beta$ be undefined.

Theorem 5.1 *If α, β are producible assemblies that are consistent and* dom $\alpha \cap$ dom $\beta \neq \emptyset$, *then $\alpha \cup \beta$ is producible. Furthermore, $\alpha \to \alpha \cup \beta$, i.e., it is possible to assemble exactly α, then to assemble the missing portions of β.*

Proof If α and β are consistent and have non-empty overlap, then $\alpha \cup \beta$ is necessarily stable, since every cut of $\alpha \cup \beta$ is a superset of some cut of either α or β, which are themselves stable.

Let Υ_α and Υ_β be assembly trees for α and β, respectively. Define an assembly tree Υ for $\alpha \cup \beta$ by the following construction. Let l_1 be a leaf in Υ_α and let l_2 be a leaf in Υ_β representing the same position $x \in$ dom $\alpha \cap$ dom β, as shown in Fig. 2a. Remove l_2 and replace it with the entire tree Υ_α. Call the resulting tree Υ'. At this point, Υ' is not an assembly tree if α and β overlapped on more than one point, because every position in dom $\alpha \cap$ dom $\beta \setminus \{x\}$ has duplicated leaves. Therefore the tree Υ' is not a hierarchical division of the set dom $\alpha \cup$ dom β, since not all unions represented by each internal node are disjoint unions. However, each node does represent a stable assembly that is the union of the (possibly overlapping) assemblies represented by its two child nodes. We will show how to modify Υ' to eliminate each of these duplicates—at which point all unions represented by internal nodes will again be disjoint—while maintaining the invariant that each internal node represents a stable assembly, proving there is an assembly tree Υ for $\alpha \cup \beta$. Furthermore, the subtree Υ_α that was placed under p_2 will not change as a result of these modifications, which implies $\alpha \to \alpha \cup \beta$.

The process to eliminate one pair of duplicate leaves is shown in Fig. 2b. Let l_1 and l_2 be two leaves representing the same point in dom $\alpha \cap$ dom β, and let a be their least common ancestor in Υ, noting that a is not contained in Υ_α since l_2 is not contained in Υ_α. Let p_a be the parent of a. Let r_1 be the root of the subtree under a containing l_1. Let r_2 be the root of the subtree under a containing l_2. Let p_2 be the parent of l_2. Remove the leaf l_2 and the node a. Set the parent of r_1 to be p_2. Set the parent of r_2 to be p_a.

Since we have replaced the leaf l_2 with a subtree containing the leaf l_1, the subtree rooted at r_1 is an assembly containing the tile represented by l_2, in the same position. Since the original attachment of l_2 to its sibling was stable, by monotonicity, the attachment represented by p_2 is still legal. The removal of a is simply to maintain that Υ is a full binary tree; leaving it would mean that it represents a superfluous "attachment" of the assembly r_2 to $\emptyset$. However, it is now legal for r_2 to be a direct child of p_a, since r_2 (due to the insertion of the entire r_1 subtree beneath a descendant of r_2, again by monotonicity) now has all the tiles necessary for its attachment to the old sibling of a to be stable. Since a was not contained in Υ_α, the subtree Υ_α has not been altered.

This process is iterated for all duplicate leaves. When all duplicates have been removed, Υ is a valid assembly tree with root $\alpha \cup \beta$. Since Υ contains Υ_α as a subtree, $\alpha \to \alpha \cup \beta$. □

It is worthwhile to observe that Theorem 5.1 does not immediately follow from Theorem 3.2. Theorem 3.2 implies that *if $\alpha \cup \beta$ is producible*, then this can be verified simply by attaching subassemblies until $\alpha \cup \beta$ is produced. Furthermore, since the hypothesis of Theorem 5.1 implies that α is producible, the greedy algorithm of Theorem 3.2 could potentially assemble α along the way to assembling $\alpha \cup \beta$, which implies that *if $\alpha \cup \beta$ is producible*, then it is producible from α. However, nothing in Theorem 3.2 guarantees that $\alpha \cup \beta$ is producible in the first place. There may be some additional details that could be added to the proof of Theorem 3.2 that would cause it to imply Theorem 5.1, but those details are likely to resemble the existing proof of Theorem 5.1, and it is conceptually cleaner to keep the two proofs separate.

Acknowledgments The author is very grateful to Ho-Lin Chen, David Soloveichik, Damien Woods, Matt Patitz, Scott Summers, Robbie Schweller, Ján Maňuch, Ladislav Stacho, Andrew Winslow for many insightful discussions, and to anonymous reviewers for their detailed and useful comments. The author was supported by NSF Grants CCF-1219274 and CCF-1162589 and the Molecular Programming Project under NSF Grants 0832824 and 1317694 and by a Computing Innovation Fellowship under NSF Grant 1019343.

References

Abel Z, Benbernou N, Damian M, Demaine ED, Demaine ML, Flatland R, Kominers S, Schweller R (2010) Shape replication through self-assembly and RNase enzymes. In: SODA 2010: proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms, Austin, Texas. Society for Industrial and Applied Mathematics

Adleman LM, Cheng Q, Goel A, Huang MDA, Kempe D, de Espanés PM, Rothemund PWK (2002) Combinatorial optimization problems in self-assembly. In: STOC 2002: proceedings of the thirty-fourth annual ACM symposium on theory of computing, pp 23–32 (2002)

Aggarwal G, Cheng Q, Goldwasser MH, Kao M-Y, de Espanés PM, Schweller RT (2005) Complexities for generalized models of self-assembly. SIAM J Comput 34:1493–1515 **Preliminary version appeared in SODA 2004**

Barish RD, Schulman R, Rothemund PWK, Winfree E (2009) An information-bearing seed for nucleating algorithmic self-assembly. Proc Nat Acad Sci 106(15):6054–6059

Cannon S, Demaine ED, Demaine ML, Eisenstat S, Patitz MJ, Schweller RT, Summers SM. Winslow A (2013) Two hands are better than one (up to constant factors). In: STACS 2013: proceedings of the thirtieth international symposium on theoretical aspects of computer science, pp 172–184

Chen H-L, Doty D (2012) Parallelism and time in hierarchical self-assembly. In: SODA 2012: proceedings of the 23rd annual ACM-SIAM symposium on discrete algorithms, pp 1163–1182

Chen H-L, Doty D, Maňuch J, Rafiey A, Stacho L (2015, to appear) Pattern overlap implies runaway growth in hierarchical tile systems. In: SoCG 2015: proceedings of the 31st international symposium on computational geometry

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT Press, Cambridge

Demaine ED, Demaine ML, Fekete SP, Ishaque M, Rafalin E, Schweller RT, Souvaine DL (2008) Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. Nat Comput 7(3):347–370 **Preliminary version appeared in DNA (2007)**

Demaine ED, Eisenstat S, Ishaque M, Winslow A (2013) One-dimensional staged self-assembly. Nat Comput 12(2):1–12 **Preliminary version appeared in DNA 2011**

Demaine ED, Patitz MJ, Rogers T, Schweller RT, Summers SM, Woods D (2013) The two-handed tile assembly model is not intrinsically universal. In: ICALP 2013: proceedings of the 40th international colloquium on automata, languages and programming, July 2013

Demaine ED, Patitz MJ, Schweller RT, Summers SM (2011) Self-assembly of arbitrary shapes using RNase enzymes: meeting the Kolmogorov bound with small scale factor. In: STACS 2011: proceedings of the 28th international symposium on theoretical aspects of computer science

Doty D (2012) Theory of algorithmic self-assembly. Commun ACM 55(12):78–88

Doty D, Lutz Jack H, Patitz MJ, Schweller RT, Summers SM, Woods D (2012) The tile assembly model is intrinsically universal. In: FOCS 2012: proceedings of the 53rd annual IEEE symposium on foundations of computer science, pp 302–310

Doty D, Patitz MJ, Reishus D, Schweller RT, Summers SM (2010) Strong fault-tolerance for self-assembly with fuzzy temperature. In: FOCS 2010: proceedings of the 51st annual IEEE symposium on foundations of computer science, pp 417–426

Fu B, Patitz MJ, Schweller RT, Sheline R (2012) Self-assembly with geometric tiles. In: ICALP 2012: proceedings of the 39th international colloquium on automata, languages and programming, pp 714–725, July 2012

Goto K, Hinob Y, Kawashima T, Kaminagab M, Yanob E, Yamamotob G, Takagic N, Nagasec S (2000) Synthesis and crystal structure of a stable S-nitrosothiol bearing a novel steric protection group and of the corresponding S-nitrothiol. Tetrahedron Lett 41(44):8479–8483

Heller W, Pugh TL (1954) "Steric protection" of hydrophobic colloidal particles by adsorption of flexible macromolecules. J Chem Phys 22(10):1778

Heller W, Pugh TL (1960) "Steric" stabilization of colloidal solutions by adsorption of flexible macromolecules. J Polym Sci 47(149):203–217

Hopcroft J, Tarjan R (1973) Algorithm 447: efficient algorithms for graph manipulation. Commun ACM 16(6):372–378

Lathrop JI, Lutz JH, Summers SM (2009) Strict self-assembly of discrete Sierpinski triangles. Theor Comput Sci 410:384–405 **Preliminary version appeared in CiE (2007)**

Luhrs C (2010) Polyomino-safe DNA self-assembly via block replacement. Nat Comput 9(1):97–109 **Preliminary version appeared in DNA 2008**

Patitz MJ (2012) An introduction to tile-based self-assembly. In: UCNC 2012: proceedings of the 11th international conference on unconventional computation and natural computation. Springer, Berlin, pp 34–62

Patitz MJ, Summers SM (2012) Identifying shapes using self-assembly. Algorithmica 64(3):481–510 **Preliminary version appeared in ISAAC (2010)**

Rothemund PWK (2001) Theory and experiments in algorithmic self-assembly. PhD thesis, University of Southern California, December 2001

Rothemund PWK, Winfree E (2000) The program-size complexity of self-assembled squares (extended abstract). In: STOC 2000: proceedings of the thirty-second annual ACM symposium on theory of computing, pp 459–468

Schulman R, Winfree E (2007) Synthesis of crystals with a programmable kinetic barrier to nucleation. Proc Nat Acad Sci 104(39):15236–15241

Schulman R, Winfree E (2009) Programmable control of nucleation for algorithmic self-assembly. SIAM J Comput 39(4):1581–1616 **Preliminary version appeared in DNA 2004**

Schweller RT, Sherman M (2013) Fuel efficient computation in passive self-assembly. In: SODA 2013: proceedings of the 24th annual ACM-SIAM symposium on discrete algorithms, pp 1513–1525, January (2013)

Wade LG (1991) Organic chemistry. Prentice Hall, Englewood Cliffs

Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology, June 1998

Winfree E (1998) Simulations of computing by self-assembly. Technical Report CaltechCSTR:1998.22, California Institute of Technology

Winfree E (2006) Self-healing tile sets. In: Chen J, Jonoska N, Rozenberg G (eds) Nanotechnology: science and computation, natural computing series. Springer, Berlin, pp 55–78

Winfree E, Liu F, Wenzler LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394(6693):539–44

Winslow A (2013) Staged self-assembly and polyomino context-free grammars. In: DNA 2013: proceedings of the 19th international meeting on DNA computing and molecular programming