Building Discrete Self-Similar Fractals in Seeded Tile Automata*

Ryan Knobel† Adrian Salinas† Robert Schweller† Tim Wylie†

Abstract

In this paper, we show that a special class of discrete self-similar fractals is *strictly* self-assembled (without error) in the seeded growth-only (no detachments) Tile Automata model. Additionally, we show that under a more restrictive version of the problem, the same class of discrete self-similar fractals is also *super-strictly* buildable– where there is the added requirement of reaching certain intermediate assemblies as the assembly grows. This contrasts with known impossibility results for the abstract Tile Assembly Model, paving the way for future work in strictly self-assembling any generalized discrete self-similar fractal.

1 Introduction

The essence of many organisms and processes of nature can often be described as a collection of simpler, self-organizing components working together to form more complex structures. The study of such mechanisms has resulted in numerous advances in designing artificial programmable systems that accomplish similar tasks. In [12], Winfree introduced the abstract Tile Assembly Model (aTAM), in which single non-rotating 'tiles' attach to growing structures. Other extensions to this model include the 2-Handed Assembly Model (2HAM) [8], where two assemblies are allowed to attach; the Signal-passing Tile Assembly model (STAM) [7], where glues can turn 'on' and 'off' and assemblies can detach; and the seeded Tile Automata Model (seeded TA) [1], where single tiles attach to a base assembly (seed) and adjacent tiles are allowed to change states. While mostly theoretical, experiments realized in the aTAM prove the potential of these programmable systems to build complex structures [10, 11, 12].

Despite varying nuances between models, building precise shapes remains a fundamental task. In particular, one of the most well-studied problems among these models is the self-assembly of self-similar fractals. In [6, 9], it was shown that the aTAM can not strictly (without error) build certain types of self-similar fractals. However, in other models, this does not hold true. In [3], it was shown that the 2HAM can finitely self-assemble a scaled-up Sierpinski carpet, while [5] showed that the 2HAM can finitely self-assemble a larger class of discrete self-similar fractals. In [7], it was shown that the Sierpinski triangle could strictly self-assemble in the STAM if tile detachments are allowed, with [4] providing constructions for any arbitrary discrete self-similar fractal with such detachments, while without such detachments, the finite number of times a STAM tile can change state makes some fractals impossible to build. The STAM is also capable of simulating Tile Automata [2] meaning these results can be ported to the STAM, however, the simulation uses detachments, which is a known result.

In this paper, we focus on building fractals in the seeded TA model without tile detachment, a model differing from the aTAM by the ability for adjacent tiles to transition states. Particularly, we show that a special class of discrete self-similar fractals can be super-strictly built (a more restricted version of strict), leaving a full treatment for future work. Super-strict assembly of a fractal essentially requires that each stage of the fractal be built in order on the way to building the infinite fractal. We feel this is a natural property to strive for as it implies that any intermediate stage of the assembly process would represent precisely the transition between two consecutive stages of the fractal, whereas without, an intermediate assembly could potentially contain a mishmash of many different incomplete fractal stages.

2 Preliminaries

This section defines the model, discrete self-similar fractals, and strictly building shapes as defined in [1, 9].

Seeded Tile Automata. Let Σ denote a set of *states* or symbols. A tile $t = (\sigma, p)$ is a non-rotatable unit square placed at point $p \in \mathbb{Z}^2$ and has a state of $\sigma \in \Sigma$. An *affinity function* Π over a set of states Σ takes an ordered pair of states $(\sigma_1, \sigma_2) \in \Sigma \times \Sigma$ and an orientation $d \in D$, where $D = \{\perp, \vdash\}$, and outputs an element of $\mathbb{Z}^{0+}$. The orientation d is the relative position to each other with $\perp$ meaning vertical and $\vdash$ meaning horizontal, with the σ_1 being the west or north state respectively. A *transition rule* consists of two ordered pairs of states $(\sigma_1, \sigma_2), (\sigma_3, \sigma_4)$ and an orientation $d \in D$, where $D = \{\perp, \vdash\}$. This denotes that if the states (σ_1, σ_2) are next to each other in orientation d (σ_1 as the west/north state) they may be replaced by the states (σ_3, σ_4). An *assembly* A is a set of tiles with states in Σ such that for

*This research was supported in part by National Science Foundation Grant CCF-2329918.

†Department of Computer Science, University of Texas Rio Grande Valley

every pair of tiles $t_1 = (\sigma_1, p_1), t_2 = (\sigma_2, p_2), p1 \neq p2$. Informally, each position contains at most one tile.

Let $B_G(A)$ be the bond graph formed by taking a node for each tile in A and adding an edge between neighboring tiles $t_1 = (\sigma_1, p_1)$ and $t_2 = (\sigma_2, p_2)$ with a weight equal to $\Pi(\sigma_1, \sigma_2)$. We say an assembly A is $\tau-$stable for some $\tau \in \mathbb{Z}^0$ if the minimum cut through $B_G(A)$ is greater than or equal to τ.

A *Seeded Tile Automata* system is a $6-$tuple $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, s, \tau)$ where Σ is a set of states, $\Lambda \subseteq \Sigma$ a set of initial states, Π is an affinity function, Δ is a set of transition rules, s is a stable assembly called the seed assembly, and τ is the temperature (or threshold). A tile $t = (\sigma, p)$ may attach to an assembly A at temperature τ to build an assembly $A' = A \cup t$ if A' is $\tau-$stable and $\sigma \in \Lambda$. We denote this as $A \rightarrow_{\Lambda,\tau} A'$. An assembly A can transition to an assembly A' if there exist two neighboring tiles $t_1 = (\sigma_1, p_1), t_2 = (\sigma_2, p_2) \in A$ (where t_1 is the west or north tile) such that there exists a transition rule in Δ with the first pair being (σ_1, σ_2), the second pair being some pair of states (σ_3, σ_4) such that $A' = (A \setminus \{t_1, t_2\}) \cup \{t_3 = (\sigma_3, p_1), t_4 = (\sigma_4, p_2)\}$. We denote this as $A \rightarrow_\Delta A'$. For this paper, we focus on systems of temperature $\tau = 1$, and all bond strengths are equal to 0 or 1.

An assembly sequence $\overrightarrow{\alpha} = \{\alpha_0, \alpha_1, \ldots\}$ in Γ is a (finite or infinite) sequence of assemblies such that each $\alpha_i \rightarrow_{\Lambda,\tau} \alpha_{i+1}$ or $\alpha_i \rightarrow_\Delta \alpha_{i+1}$. An assembly subsequence $\beta = \{\alpha'_0, \alpha'_1, \ldots\}$ in Γ is a (finite or infinite) sequence of assemblies such that for each α'_i, α'_{i+1} there exists an assembly sequence $\overrightarrow{\alpha} = \{\alpha'_i, \ldots, \alpha'_{i+1}\}$.

We define the *shape* of an assembly A, denoted $(A)_\Lambda$, as the set of points $(A)_\Lambda = \{p | (\sigma, p) \in A\}$.

Discrete Self-Similar Fractals. Let $1 < c, d \in \mathbb{N}$ and $X \subseteq \mathbb{N}^2$. We say that X is a $(c \times d)$-discrete self-similar fractal if there is a set $G \subseteq \{0, \ldots, c-1\} \times \{0, \ldots, d-1\}$ with $(0,0) \in G$, such that $X = \bigcup_{i=1}^{\infty} X_i$, where X_i is the i^{th} *stage* of G satisfying $X_0 = \{(0,0)\}$, $X_1 = G$, and $X_{i+1} = \{(a,b) + (c^i v, d^i u) | (a,b) \in X_i, (v,u) \in G\}$. In this case, we say that G *generates* X. We say that X is a discrete self-similar fractal if it is a $(c \times d)$-discrete self-similar fractal for some $c, d \in \mathbb{N}$. A generator G is termed *feasible* if it is a connected set, and there exist (not necessarily distinct) points $(0, y), (c-1, y), (x, 0), (x, d-1) \in G$, i.e., a pair of points on each opposing edge of the generator bounding box that share the same row or column. Note that the fractal generated by a generator is connected if and only if the generator is feasible. For the remainder of this paper we only consider feasible generators.

Strict and Super-strict. Let X be a discrete self-similar fractal with feasible generator G. Consider a seeded TA system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, s, \tau)$ with $(s)_\Lambda = G$, and let S denote the set of all valid assembly sequences

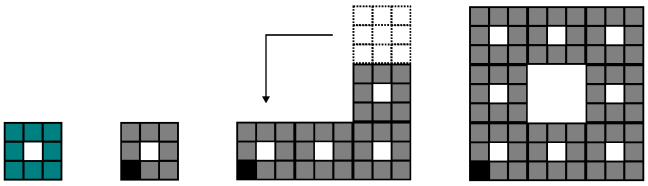

Figure 1: From left to right: the generator, the seed assembly (with the tile in black representing the origin tile), the assembly at the start of step 4 and the assembly at stage 2 (or the end of stage 1).

for Γ. Γ *strictly* builds X if $\forall \overrightarrow{\alpha_i} = \{s, \alpha_1, \ldots, \alpha_i, \ldots\} \in S$, $\overrightarrow{\alpha_i}$ is infinite and $\lim_{i \to \infty} (\alpha_i)_\Lambda = X$. We further say that Γ *super-strictly* builds discrete self-similar fractal X if $\forall \overrightarrow{\alpha_i} \in S$, there exists a subsequence $\beta = \{s, \alpha'_1, \ldots\}$ of $\overrightarrow{\alpha_i}$ such that each $(\alpha'_i)_\Lambda = X_i$.

Other Definitions. Let G be a feasible generator with corresponding points $(0, y), (c-1, y), (x, 0), (x, d-1) \in G$, X be the discrete self-similar fractal corresponding to G, and A be an assembly such that $(A)_\Lambda = X_i$ for some $i \in \{1, 2, \ldots\}$. We denote *key positions* as four points $p_N, p_E, p_W, p_S \in (A)_\Lambda$ satisfying $p_N = (x + c^{i-1} \cdot x, d^i - 1)$, $p_E = (c^i - 1, y + d^{i-1} \cdot y)$, $p_W = (0, y + y \cdot d^{i-1})$ and $p_S = (x + c^{i-1} \cdot x, 0)$. The four tiles $t_N, t_E, t_W, t_S \in A$ with positions p_N, p_E, p_W, p_S, respectively, are called *key tiles*. We denote $t_0 \in G$ the *origin tile* if t_0 has position $(0,0)$.

Let $G_G = (V, E)$ be the embedded graph formed by adding a vertex for each point $p \in G$ and adding an edge between vertices representing neighboring points $p_1, p_2 \in G$. Let $H = \langle h_0, \ldots, h_m \rangle$ $(m = |G| - 1)$ denote a Hamiltonian path in G_G, and let vertex h_0 represent the origin of the generator, where each h_j represents $p_j = (w_j, u_j)$. Given X_i, the i^{th} stage of generator G, denote $X_i^j = \{(a + c^i w_j, b + d^i u_j) \mid (a,b) \in X_i, j \in \{0, \ldots, m\}\}$, where j is the j^{th} *step* for stage i.

Additionally, we denote a particular assembly A as A_i if $(A)_\Lambda = X_i$ and A as A_i^J if $(A)_\Lambda = \bigcup_{k=0}^{J} X_i^k$, where $J \in \{0, \ldots, m\}$. To refer to a specific sub-assembly of A_i^J corresponding to step $j \in \{0, \ldots, J\}$ for stage i, we use A_i^j, where $A_i^j \subset A_i^J$ and $(A_i^j)_\Lambda = X_i^j$.

3 Construction

Given a feasible generator G where the resulting embedded graph G_G has a Hamiltonian path, we construct a seeded TA system with seed s ($(s)_\Lambda = G$) and origin tile t_0 that super-strictly builds the corresponding fractal infinitely. To start, the number of points m in the generator excluding the origin determines the number of steps m needs to scale the assembly from stage i to stage $i+1$. We denote each point in the generator as p_j, where j represents the distance from itself and the origin p_0 following a selected Hamiltonian path P in G_G.

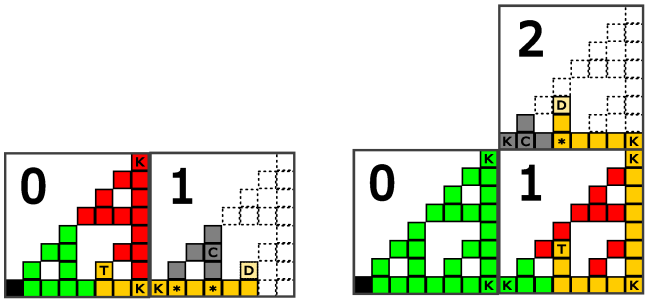

Figure 2: The Sierpinski triangle starting from stage 3 with $m = 2$ steps. The tile marked T is sending a signal (yellow) to place itself at position D. Tiles marked K are key tiles (origin tile is also a key tile, just not labeled). Tiles marked $*$ have a cap in the direction of the gray placed tiles. Tiles marked C had 2 caps, so the cap shifted to the tiles marked $*$.

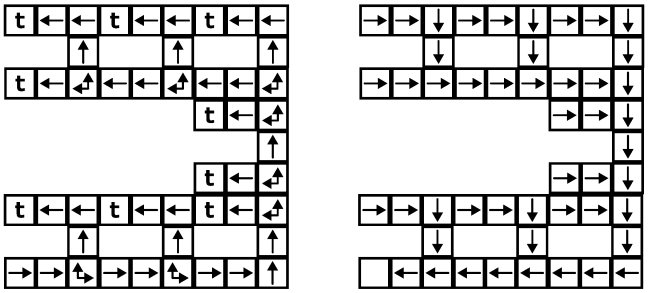

Figure 3: An example of the 'next' (left) and 'previous' (right) directions for each tile. Tiles marked t are terminal tiles, meaning they have no 'next' direction. The only tile without a 'previous' direction is the origin tile, which is the tile located at the bottom left. Note that the 'next' and 'previous' directions at each tile do not always include all adjacent tiles.

If we let $d(p_j, p_{j-1})$ denote the relative position of p_j to p_{j-1} (north, east, west or south), then we can represent the sequence of directions the assembly will grow.

The high-level idea of the construction is as follows: given an initial assembly A^0, translate a copy of A^0 in the direction of $d(p_1, p_0)$ and denote the copy as A^1. Repeat for all A^j, copying A^j in direction $d(p_{j+1}, p_j)$ until $j = m$. Set $A' = \bigcup_{j=0}^{m} A^j$, and then continue the process with $A^0 = A'$. See Figure 1 for an example.

However, since the seeded TA model is limited to single attachments and transitions, a direct implementation of this high-level idea is not possible. Instead, we give each tile the responsibility of placing itself in the correct location, with the final result being a copied translation from A^j to A^{j+1}. Thus, a crucial part of our construction is the ability to store information and send signals through the assembly. This section focuses on describing each of these components more thoroughly.

3.1 Storing Information

In order to correctly copy the base assembly, every tile needs specific information. This information can implicitly be stored as the state σ of the tile.

Current State ($STATE(t)$)**.** As each tile is responsible for placing itself in the right location at the next step, it is important to know whether each tile has either 1) not placed itself yet, 2) is currently placing itself or 3) has already placed itself. $STATE(t)$ denotes the current state of tile t. In Figure 2, green tiles are tiles with $STATE(t) =$ complete, red tiles are tiles with $STATE(t) =$ incomplete and the yellow tile marked T has $STATE(t) =$ waiting. Gray tiles are tiles that have been placed from the current step, so they must wait until the current step finishes.

Direction to Key Tiles ($KEY_d(t)$)**.** With the key tile information, signals are sent in the direction of the correct key tile. For instance, if the assembly is being copied to the north at step j, signals are sent in the direction of $t_N^j \in A_i^j$. $KEY_d(t)$ denotes this direction for t. To reference all 4 key tiles, we use $KEY_{NEWS}(t)$. This is illustrated in Figure 5a.

Next/Previous Tiles ($NEXT(t)/PREV(t)$)**.** This serves the purpose of knowing where each tile's neighbors are (or should be). $NEXT(t)$ denotes the direction to the 'next' tiles from t, which usually signifies which directions the signal can propagate, excluding the source direction. $PREV(t)$ denotes the direction to the previous tile from t, which usually signifies the direction a signal comes from. We use $NEXT_t(t)$ to denote the tile adjacent to t in direction $NEXT(t)$ (or similarly, the set of tiles adjacent to t for each direction in $NEXT(t)$). Similarly, $PREV_t(t)$ denotes the tile adjacent to t in direction $PREV(t)$. This is described in Figure 3.

The State of Neighboring Sub-assemblies ($SUB_d(t)$)**.** This is crucial for several reasons. Firstly, this creates the order in which tiles are placed. Secondly, this makes it possible to keep track of which direction the signal is coming from, and once the tile is placed, where the signal needs to return to. $SUB_d(t)$ denotes the state of the sub-assembly (whether all tiles have been placed or not) stemming from the neighboring tile of t in direction d. To reference the state of all sub-assemblies adjacent to t, we use $SUB_{NEWS}(t)$. Additionally, we refer to a specific sub-assembly as $SUBASM_d(t)$, denoting the sub-assembly stemming from tile t in direction d. See Figure 5b for an example.

The Tile being Transferred ($TRANS(t)$)**.** To distinguish between different signals, each tile keeps track of which tile the signal started from. $TRANS(t)$ is used to denote the tile that the signal is coming from.

Step. Each tile stores which step it is a part of. This allows an assembly A_i^J to know which tiles to use (tiles in A_i^j) to create sub-assembly A_i^{j+1}. Additionally, A_i^j will only send signals to $t^j_{d(p_{j+1},p_j)}$.

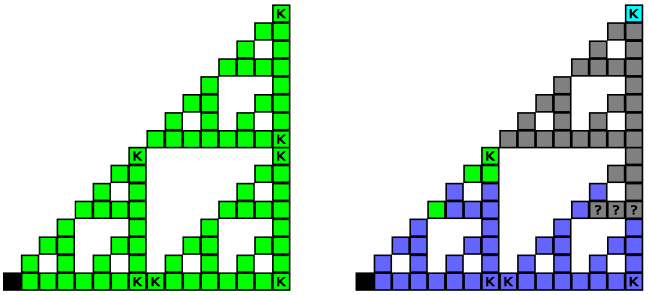

Figure 4: The Sierpinski triangle resetting at the end of stage 3. Tiles marked ? are waiting for the sub-assemblies adjacent to reset (the blue tiles to the north and west). Gray tiles have been reset. Blue tiles are transmitting the reset signal. Light blue tiles marked 'K' are the new key tiles for the assembly.

Terminal ($TERM(t)$)**.** Tiles must know when they are at the end of a sub-assembly. Once a terminal tile is placed, the system knows part of the sub-assembly is complete. $TERM(t)$ is a boolean that denotes whether tile t is terminal or not. In Figure 3, these tiles are marked t.

Caps ($CAP_d(t)$)**.** Copied assemblies require one extra piece of information. As the shape grows, there are points where signals can branch in multiple directions. To direct this, signals always go 'left' when there is a fork. If the sub-assembly in this direction is already constructed, a cap is placed to prevent signals from going in that direction, and it instead goes to the next path. If all paths have a cap, then it turns around and the cap is shifted to reflect that all paths are complete. Caps start from terminal tiles and gradually shift as the sub-assemblies are completed. $CAP_d(t)$ is a boolean that denotes whether tile t has a cap in direction d. Figure 2 shows an example of how caps are used and shifted.

3.2 Signal Passing

In addition to the stored information, it is important that tiles can communicate through signal passing. This is done via transition rules.

Tile Placement Ordering. The order in which tiles place themselves follows a 'left' first order (described in Section 3.3). As the tiles place themselves and are marked complete, transition rules prompt the next tile to start placing itself.

Tile Placement Signals. When a tile t_i^j is placing itself, the signal is transmitted from t_i^j to the tile adjacent to the target position for t_i^j. Transition rules make this possible by transferring the signal between adjacent tiles. In Figure 2, the transmission of this placement signal is represented as the sequence of yellow tiles.

Tile Placement Completion Signals. Once the tile is placed in the correct location, a 'completion' signal gets sent back the same direction as the placement signal. Once this signal reaches the tile getting placed from A_i^j, the tile is marked as complete.

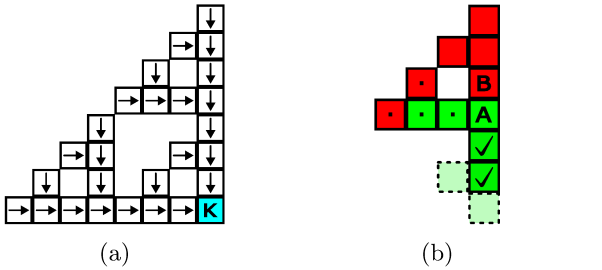

Figure 5: (a) An example of the direction stored at each tile for t_E, the tile marked K. (b) An example of how sub-assemblies work. The check mark denotes the sub-assemblies to the south of tile A are completed. The sub-assemblies to the west and north of A, however, are not. As a result, $SUB_S(A)$ = complete, $SUB_W(A)$ = incomplete and $SUB_N(A)$ = incomplete. Tiles marked with . are part of $SUBASM_W(A)$. Note that B does not start placing itself until $SUB_W(A)$ and $SUB_S(A)$ are both marked completed.

Cap Signals. When a terminal tile is placed, as the 'completion' signal gets sent back in the sub-assembly being created, a 'cap' is sent back with it to mark the sub-assembly as complete. This forces future signals to go a different path to complete a different sub-assembly.

Reset Signals When a stage is completed, reset signals are sent to update current state, direction to key tiles, state of neighboring tiles and step, as well as removing any remaining caps. Figure 4 illustrates the resetting process.

Figure 2 details the construction outlined in Sections 3.1 and 3.2. The following section provides more specific details to express how the system interacts to create these fractals.

3.3 Approach

This section describes the process for taking an assembly A_i to A_{i+1}, assuming G is a feasible generator for $(A_i)_\Lambda$, $H = \langle h_0, \ldots, h_m \rangle$ is a Hamiltonian path in G_G starting from the origin where m is the number of steps, i is the stage and $t_0 \in A_i^0$ is the origin tile. Additionally, we denote t_d^j as the key tile for direction d in assembly A_i^j and $d_j = d(p_{j+1}, p_j)$. We briefly define some additional terminology:

$OPP(d)$**.** This denotes the complement of direction d, e.g., $OPP(north) = south$.

$LEFT(D)$**.** Consider $D \in \{\{N\}, \{E\}, \{W\}, \{S\}, \{N, E\}, \{E, S\}, \{N, W\}, \{W, S\}\}$, where N, E, W, S represent north, east, west and south respectively. $LEFT(D)$ denotes the 'left' direction for D. This is 1) North if $D = \{N, E\}$ or $\{N\}$, 2) East if $D = \{E, S\}$ or $\{E\}$, 3) West if $D = \{N, W\}$ or $\{W\}$ and 4) South if $D = \{W, S\}$ or $\{S\}$.

Conversely, $RIGHT(D)$ denotes $NEXT(D) \setminus LEFT(D)$. For a tile t, we use $LEFT_t(D)/RIGHT_t(D)$

to denote the tile adjacent to t in direction $LEFT(D)/RIGHT(D)$, respectively.

RESET(t)**.** At the end of stage i, tiles must reset. This includes 1) updating the direction to the new 4 key tiles and 2) updating the 'next' direction for former key tiles (see Figure 6). $RESET(t)$ denotes tile t resetting, defined as follows:

- If $t = t^j_{d_j}$, where $p_j \in X_1$ is the key position for d_j, set $KEY_{d_j}(t) =$ current tile.
- For each tile $t_a \in NEXT_t(t)$ (if all t_a have reset) set each $KEY_d(t) = PREV(t)$ if $KEY_d(t_a) = d(t, t_a)$. If there is a tile such that $KEY_d(t_a) \neq d(t, t_a)$, set $KEY_d(t) = KEY_d(t_a)$. If t_a is a key tile for d, set $KEY_d(t) = d(t_a, t)$.
- Clear $TERM(t_c)$ and update $NEXT(t_c)$ if appropriate.

3.3.1 Algorithm

Now we describe the algorithm. For better comprehension, we describe the algorithm using sub-processes. Technical descriptions of these sub-processes are included in Section 6.

Start with $j = 0$ and let $t_c = t_0$ denote the current tile getting placed, starting with the origin tile. Let A_i^{j+1} represent the translated assembly being created at step $j + 1$. The following will be repeated until $j = m$. While $SUB_{PREV(t^j_{d_j})}(t^j_{d_j})$ and $SUB_{NEXT(t^j_{d_j})}(t^j_{d_j})$ are not marked as completed:

1. Let t_a denote the tile adjacent to t_c in direction $KEY_{d_j}(t_c)$. Run *send_placement_signal*$(t_c, t^j_{OPP(d_j)}, t^j_{d_j}, t_a, d_j)$ to send a signal through the assembly to place t_c in the correct location.
2. The placement signal stops at the tile adjacent to the target position by always traversing 'left' until a tile no longer exists. Run *place_tile*(t_a, t'_c, p) to place the tile at this location, where t_a is the tile adjacent to position p, the target position for t_c. This places t_c in the correct location as t'_c.
3. Retrace the signal to the tile that got placed by running *send_completion_signal*(t'_c, t_a, d_j). This also marks the tile as complete.
4. Mark sub-assemblies as complete if needed. Run *mark_completed_sub-assemblies*(t_c, t_a) if $TERM(t_c) =$ True, where t_c is the tile that just placed itself and t_a is the tile adjacent to t_c such that $STATE(t_a) =$ complete.
5. Choose the next tile to be placed. Let t_c denote the last tile updated and C be the tiles in $NEXT_{t_c}(t_c) \cup PREV_{t_c}(t_c)$ that have a completed state. If $t_c \neq t^j_{d_j}$, repeat from (1) with the new $t_c = LEFT_{t_c}(PREV(t_c) \cup NEXT(t_c) \setminus C)$.

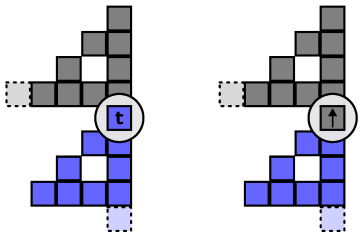

Figure 6: The highlighted tile is initially set as terminal. Since the tile used to be a key tile, resetting also updates the 'next' direction if appropriate.

6. If $t_c = t^j_{d_j}$, $j \neq m$ (the stage is not yet completed) and $SUB_{PREV(t^j_{d_j})}(t^j_{d_j})$ and $SUB_{NEXT(t^j_{d_j})}(t^j_{d_j})$ are marked as completed (every tile has now been placed), run *start_next_step*$(t^{j+1}_{OPP(d_j)}, t^{j+1}_{d_{j+1}}, d_j, d_{j+1})$ to signal for the next sub-assembly to start being created. Repeat from (1) with $j = j + 1$, $A_i^j = A_i^{j+1}$ and clear $TRANS(t_c)$.
7. If instead $j = m$, the initial assembly has now been up-scaled and has reached the end of stage i. To repeat this process, the assembly now has to reset. Run *reset*$(t^m_{OPP(d_{m-1})})$.
8. Repeat the algorithm.

A primary reason as to why this algorithm works is the existence of a Hamiltonian path in the generator, as this dictates the directions in which the fractal grows. This allows growth of the fractal for any step to only depend on the created sub-assembly from the previous step, regardless of whether or not the resulting assembly contains a Hamiltonian path or not. If a generator does not contain a Hamiltonian path, then some sub-assemblies of the fractal must be used multiple times to create copies in multiple directions, which results in synchronicity issues as multiple signals could exist in the assembly at once.

4 Results

We now show that any feasible generator G with a Hamiltonian path in G_G can be super-strictly built by a seeded TA system Γ. Let G be a feasible generator for discrete self-similar fractal X, with $H = \langle h_0, \ldots, h_m \rangle$ denoting a Hamiltonian path in G_G such that each h_j corresponds to point $p_j = (w_j, u_j) \in G$. Let $d_j = d(p_{j+1}, p_j)$, A be the current assembly starting from $A = A_i^J$ for some $J \in \{0, \ldots, m-1\}$, and $A_i^{j+1'} = A \setminus A_i^J$. We denote the copy of a tile t as t'.

Lemma 1 *Under the construction from Section 3, tile $t^j_{OPP(d_j)} \in A_i^j$ must be the first tile to place itself.*

Proof. This is due to geometry. While there may be other adjacent tiles between A_i^j and the sub-assembly

being created, $A_i^{j+1'}$, $t_{d_j}^j$ is the only tile that recognizes the existence of $A_i^{j+1'}$. Thus, the only way to send a signal to $A_i^{j+1'}$ is through $t_{d_j}^j$, and the only adjacent tile to $t_{d_j}^j$ in $A_i^{j+1'}$ is $t_{OPP(d_j)}^{j'}$. □

Lemma 2 *Let $t_i = (\sigma_i, p_i) \in A_i^j$ and let $t_f = (\sigma_f, p_f) \in A_i^{j+1'}$ represent the tile adjacent to p_i', the target location for t_i'. A signal will follow exactly 1 path from t_i to t_f.*

Proof. From Section 3.3, signals will always follow one path in A_i^j and $A_i^{j+1'}$. We show that this signal ends at tile t_f. This can be done by comparing the order in which tiles are chosen to be placed to the direction that the signal travels.

To prove equivalence, we show that for each tile in $A_i^{j+1'}$, $PREV(t_c) \cup NEXT(t_c) \setminus d(t_p, t_c) = NEXT(t_c^*)$. We consider 2 cases:

Case 1: $t_c^* = t_{OPP(d_j)}^{j+1}$. From Section 3.3, $NEXT(t_c^*) = NEXT(t_{OPP(d_j)}^j) \cup PREV(t_{OPP(d_j)}^j) \setminus OPP(d_j)$. Initially, t_p is the tile in direction $OPP(d_j)$ from t_c. This results in $PREV(t_c) \cup NEXT(t_c) \setminus d(t_p, t_c)$.

Case 2: t_c^* is any other tile. Let t_p^* denote the tile adjacent to t_c^* from which the signal came from, with t_p, t_c denoting the corresponding tiles from A_i^j. We consider 2 scenarios.

1. $t_c \in NEXT_{t_p}(t_p)$. From Section 3.3, $NEXT(t_c^*) = PREV(t_c) \cup NEXT(t_c) \setminus d(t_p, t_c)$.
2. $t_c \in PREV_{t_p}(t_p)$. From Section 3.3, $NEXT(t_c^*) = NEXT(t_c) \cup PREV(t_c) \setminus (d(t_p, t_c) \cup \{\textit{directions to tiles not in step } j\})$. The only time there exists a direction to a tile not in step j is when t_c is the first tile placed in step j. Since this is no longer the case for step $j+1$, we get rid of this direction from $NEXT(t_c^*)$, and since the next tile chosen to be placed from t_c does not consider this direction either, the equivalence holds. □

Lemma 3 *Let $t_i = (\sigma_i, p_i) \in A_i^j$ and let $t_f = (\sigma_f, p_f) \in A_i^{j+1'}$ represent the tile adjacent to p_i', the target location for t_i'. Tile t_f will place tile t_i' at position $p_i' = p_i + a \cdot d_j^*$, where $a - 1 \in \mathbb{N}$ represents the distance $|p_{d_j} - p_{OPP(d_j)}|$ and $d_j^* \in \{0, 1\}^2$ is a 2-D vector denoting the direction.*

Proof. Let $d_j^* = [0, 1], [1, 0], [-1, 0], [0, -1]$ represent d_j = north, east, west and south, respectively. In the case of $t_{OPP(d_j)}$' being the tile placed, the signal will stop at tile $t_{d_j}^j$ with position $p_{d_j} = p_{OPP(d_j)} + (a-1) \cdot d_j^*$. Tile $t'_{OPP(d_j)}$ is then placed at position $p'_{OPP(d_j)} = p_{d_j} + d_j^* = p_{OPP(d_j)} + a \cdot d_j^*$.

For any other tile t_i, as described in Lemma 2, we know 2 things: 1), the signal from tile t_i will stop at tile t_f adjacent to position p_i' and 2) the order in which tiles are chosen to be placed is equivalent to the direction in which signals are passed. Since the relative position of p_i' to $t'_{OPP(d_j)}$ is the same as p_i to $t_{OPP(d_j)}$ and $t'_{OPP(d_j)} = t_{OPP(d_j)} + a \cdot d_j$, it follows that $p_i' = p_i + a \cdot d_j^*$. □

Lemma 4 *Let $t_i = (\sigma_i, p_i) \in A_i^j$ and let $t_i' = (\sigma_i', p_i') \in A_i^{j+1'}$ represent the copy of t_i. A signal will follow exactly 1 path from t_i' to t_i.*

Proof. By Lemma 2, there exists one path from p_i to the tile adjacent to p_i'. Thus, when tile t_i' is placed at position p_i', the converse holds true by retracing this path. □

Theorem 5 *There is at most one tile transmitting a signal in A.*

Proof. By contradiction. Assume that there exists 2 tiles $t_1 \neq t_2$ transmitting signals through A_i^j and let $A^{j_1^*}, A^{j_2^*} \subset A_i^j$ denote 2 sets of tiles such that:

1. $\forall t_a \in A^{j_1^*}$, $STATE(t_a) =$ complete.
2. $\forall t_b \in NEXT_{t_a}(t_a) \cup PREV_{t_a}(t_a)$, $t_b \in A^{j_1^*}$, $STATE(t_b) =$ incomplete or $t_b = t_1$.

where the same applies for $A^{j_2^*}$ and t_2. We consider 2 cases:

Case 1: $A^{j_1^*} \cap A^{j_2^*} = \emptyset$. This implies $t_{OPP(d_j)}^j \in A^{j_1^*}$ or $t_{OPP(d_j)}^j \in A^{j_2^*}$, but not both. By Lemma 1, $t_{OPP(d_j)}^j$ must be the first tile placed, resulting in a contradiction.

Case 2: $A^{j_1^*} \cap A^{j_2^*} \neq \emptyset$. Consider a tile t^* such that $t_1 \in SUBASM_{d_1}(t^*)$ and $t_2 \in SUBASM_{d_2}(t^*)$. Since $SUBASM_{RIGHT(\{d_1,d_2\})}(t^*)$ must wait for $SUBASM_{LEFT(\{d_1,d_2\})}(t^*)$ to be completed, it must be that $d_1 = d_2$. This implies that $t_1 = t_2$. □

Theorem 6 *Step $j+2$ will start only when step $j+1$ is completed.*

Proof. By our construction, since $t_{d_j}^j$ is the tile communicating between A_i^j and $A_i^{j+1'}$, both $SUB_{PREV(t_{d_j})}(t_{d_j})$ and $SUB_{NEXT(t_{d_j})}(t_{d_j})$ must be marked as completed before step $j+2$ begins. This is true only when $\forall t \in A_i^j$, $STATE(t) =$ complete. □

Lemma 7 *Let $A_i^M = \bigcup_{j=0}^{m} A_i^j$ denote the resulting assembly at step m for stage i. Every tile will reset before moving to stage $i+1$.*

Proof. This is due to our construction. A tile t will only reset when $\forall t_a \in NEXT_t(t)$, t_a is reset. The only time this is not true is when t is terminal, which marks the end of a sub-assembly. □

Lemma 8 *Let $A_i^M = \bigcup_{j=0}^{m} A_i^j$ denote the resulting assembly at step m for stage i. When t_0 resets, there will exist at most 4 key tiles and $KEY_{NEWS}(t) \forall t \in A_i^M$ is updated to point to these new key tiles.*

Proof. We first show that there will exist at most four key tiles, one for each direction d. From Section 3.3, t_d must appear only in some step j. Thus, as the assembly resets, t_d^j resets as the new t_d for the up-scaled assembly, and all other $t_d^k \; \forall k \neq j \in \{0, \ldots, m\}$ are reset to normal tiles. This leaves at most four key tiles.

Next, we show that every tile will point to the direction of the new t_d's. We show this by contradiction. Assume that there exists a tile t such that $KEY_d(t) = PREV(t)$, but $t_d \in SUBASM_{NEXT(t)}(t)$. This implies that for $t_a = PREV_{t_d}(t_d)$, $KEY_d(t_a) = PREV(t_a)$, which is true only if t_d is not the key tile for direction d. Hence, $t_d \notin SUBASM_{NEXT(t)}(t)$. □

Lemma 9 *Let $A^{j*} \subseteq A_i^j$ where $\forall t \in A^{j*}$, $STATE(t) =$ complete. At the end of step $j+1$, $(A_i^{j+1'})_\Lambda = (A_i^j)_\Lambda + a \cdot d_j^*$, where $a - 1 \in \mathbb{N}$ represents the distance $|p_{d_j} - p_{OPP(d_j)}|$ and $d_j^* \in \{-1, 0, 1\}^2$ is a 2-D vector denoting the direction.*

Proof. We use Lemmas 1 and 3 to construct an inductive proof. Let $d_j^* = [0, 1], [1, 0], [-1, 0], [0, -1]$ represent $d_j =$ north, east, west and south, respectively.

Base case. $|A_i^{j*}| = 0$, with $t_1 = t_{OPP(d_j)}^j$ being the first tile copying itself (Lemma 1). By Lemma 3, $t'_{OPP(d_j)}$ is placed at position $p'_1 = p_1 + a \cdot d_j^*$.

Inductive step. $|A_i^{j*}| = k$, with t_{k+1} being the tile copying itself. By Lemma 3, t'_{k+1} is placed at position $p'_{k+1} = p_{k+1} + a \cdot d_j^*$. It holds that $(A_i^{j+1'})_\Lambda = (A^{j*})_\Lambda + a \cdot d_j^*$. Thus, $(A_i^{j+1})_\Lambda = (A_i^j)_\Lambda + a \cdot d_j^*$. □

Theorem 10 *At the end of stage i, $(A_i^M)_\Lambda = (A_{i+1})_\Lambda$.*

Proof. Follows from Lemma 9. For each A_i^K with $K \in \{0, \ldots, m-1\}$, a new sub-assembly A_i^{k+1} is constructed such that the new assembly $A_i^{K+1} = A_i^K \cup A_i^{k+1}$ satisfying $(A_i^{K+1})_\Lambda = X_i^{K+1}$. Thus, the final assembly $A_i^M = A_i^{M-1} \cup A_i^m$ with $(A_i^M)_\Lambda = \bigcup_{j=0}^{m} X_i^j = X_{i+1} = (A_{i+1})_\Lambda$. □

Theorem 11 *Let X be a discrete self-similar fractal with feasible generator G in bounding box $c \times d$ such that G_G has a Hamiltonian path $\langle h_0, \ldots, h_m \rangle$ where h_0 represents the origin. There exists a seeded TA system Γ with $O(|G|)$ states, $O(|G|^2)$ transitions and $O(|G|^2)$ affinities that super-strictly builds X.*

Proof. We start by showing Γ strictly builds X. This follows from Theorem 10. We start with seed s, where $(s)_\Lambda = G$ and each $t_j \in s$ stores $NEXT(t_j) = d(p_{j+1}, p_j)$ (if p_{j+1} exists), $PREV(t_j) = d(p_{j-1}, p_j)$ (if p_{j-1} exists) and $TERM(t_m) =$ True. Denote the assembly as A_1. By Theorem 10, applying the construction from Section 3 yields a new assembly $A_2 = \bigcup_{j=0}^{m} A_i^j$ with shape $(A_2)_\Lambda = X_2$. Repeating this for all A_i yields $\lim_{i \to \infty} (A_i)_\Lambda = X$.

Now we show that Γ super-strictly builds X. To do so, we consider 2 cases:

1. The assembly A_i^M at the end of stage i before resetting. Leading up to this point, the order in which tiles are placed and signals are passed is deterministic. As a result, there exists 1 unique valid assembly sequence from A_i to A_i^M.
2. The assembly A_{i+1} after A_i^M resets. While there no longer exists 1 unique valid assembly sequence from A_i^M to A_{i+1}, Lemmas 7 and 8 show that every tile will reset to point to the 4 new key tiles. From (1), the rest of the local information at each tile will remain the same. Thus, every valid assembly sequence from A_i^M to A_{i+1} starts with A_i^M and ends with A_{i+1}.

Choose $\beta = \{s, A_2^M, A_3^M, \ldots\}$ or $\beta = \{s, A_2, A_3, \ldots\}$. It follows that Γ super-strictly builds X.

Disregarding steps, the total number of ways information can be locally stored at any tile is $O(1)$ since a tile has at most 4 neighbors. However, as tiles need to distinguish between different sub-assemblies representing different steps, this results in $O(|G|)$ different states. Similarly, since transition rules and affinities use combinations of 2 states, this results in $O(|G|^2)$ transition rules and affinity rules. □

5 Conclusion

In this paper, we present a method to strictly build fractals infinitely under the assumption that the generator is feasible and contains a Hamiltonian path. This contrasts with previously known results from similar (but slightly differing) models such as the aTAM, where some fractals, such as the Sierpinski triangle, are shown to be impossible to build strictly. Additionally, we show that this class of fractals can be super-strictly built, as our construction guarantees stopping at unique intermediate assemblies for all possible assembly sequences, where each intermediate assembly represents a different stage of the fractal. However, there remains several open questions:

- Our construction strictly builds fractals infinitely with states linear in the size of the generator and transitions and affinities quadratic in the size of the generator. Is there an alternative method that reduces the state, transition and affinity counts?

- Is it possible to construct all fractals infinitely? If not, what fractals are impossible to build?
- Does there exist a seeded TA system that can strictly build any fractal infinitely?
- Our work focuses on systems with temperature 1. Is it possible to take advantage of systems with higher temperatures to strictly build these fractals more efficiently, or does higher temperatures increase the complexity of the problem?

6 Full Details for Algorithm

Below are the full details for the sub-processes used in the algorithm described in Section 3.3.

send_placement_signal(t_c, $t^j_{OPP(d_j)}$, $t^j_{d_j}$, t_a, d_j):

1. Set $STATE(t_c) =$ waiting.
   - If $t_c = t^j_{OPP(d_j)}$, set $NEXT(t'_c) = NEXT(t_c) \cup PREV(t_c) \backslash OPP(d_j)$, $PREV(t'_c) = OPP(d_j)$ and $TERM(t'_c) =$ False.
   - Else if $STATE(t_a) =$ complete and the number of tiles from step j in $NEXT_{t_c}(t_c) \cup PREV_{t_c}(t_c)$ is $= 1$, set $TERM(t'_c) =$ True and $PREV(t'_c) = d(t_a, t_c)$.
   - Else if $STATE(t_a) =$ complete and $t_c \in PREV_{t_a}(t_a)$, set $NEXT(t'_c) = NEXT(t_c) \cup PREV(t_c) \setminus (d(t_a, t_c) \cup \{$*directions to tiles not in step j*$\})$ and $PREV(t'_c) = d(t_a, t_c)$.
   - Else, set $NEXT(t'_c) = NEXT(t_c)$ and $PREV(t'_c) = PREV(t_c)$.
   - Set $KEY_{NEWS}(t'_c) = KEY_{NEWS}(t_c)$, $TERM(t'_c) = TERM(t_c)$ if $TERM(t'_c)$ is not defined yet, $SUB_{d(t_c,t_a)}(t_a) =$ waiting and $TRANS(t_a) = t'_c$.
   - Let $t_c = t_a$.
2. While $t_c \neq t^j_{d_j}$:
   - Set $SUB_{d(t_c,t_a)}(t_a) =$ waiting and $TRANS(t_a) = TRANS(t_c)$.
   - Let $t_c = t_a$.
3. If no tile exists adjacent to t_c in direction d_j, stop. Otherwise, set $SUB_{OPP(d_j)}(t_a) =$ waiting and $TRANS(t_a) = TRANS(t_c)$.
4. Repeat the following:
   (a) Let $t_a = LEFT_{t_c}(NEXT(t_c))$ if $!CAP_{LEFT(NEXT(t_c))}(t_c)$, else set $t_a = RIGHT_{t_c}(NEXT(t_c))$.
   (b) If t_a exists, set $SUB_{d(t_c,t_a)}(t_a) =$ waiting and $TRANS(t_a) = TRANS(t_c)$. Set $t_c = t_a$ and repeat from (a).
   (c) If t_a does not exist, stop.

place_tile(t_c, t_c', p):

1. Place t'_c in position p and set $SUB_{d(t_c,t'_c)}(t'_c) =$ maybe. If $TERM(t'_c)$, set $SUB_{d(t_c,t'_c)}(t'_c) =$ maybe with cap.

send_completion_signal(t_c, t_a, d_j):

1. Set $SUB_{d(t_a,t_c)}(t_c)$ to its original state, clearing $TRANS(t_c)$ and changing $SUB_d(t_a) =$ waiting to $SUB_d(t_a) =$ maybe for the direction d that the signal came from.
2. If $length(NEXT(t_c)) =$ number of caps on t_c, set $SUB_d(t_a) =$ maybe with cap and clear the cap from t_c. Otherwise, set $SUB_d(t_a) =$ maybe and leave the cap on t_c in direction $LEFT(t_c)$.
3. If $STATE(t_a) =$ waiting, set $SUB_{d(t_a,t_c)}(t_c)$ to its original state, clearing $TRANS(t_c)$ and changing $STATE(t_a) =$ waiting to $STATE(t_a) =$ complete. Otherwise, set $t_c = t_a$, let t_a be the tile adjacent to t_c from which the signal came from and repeat from (1).

mark_completed_sub-assemblies(t_c, t_a):

1. Repeat the following until t_a is not updated:
   (a) Set $SUB_{d(t_c,t_a)}(t_a) =$ complete.
   (b) If $length(SUB_{NEWS}(t_c) =$ complete$) = length(NEXT(t_c) \cup PREV(t_c))$, set $t_c = t_a$ and let t_a be the tile next to t_c with $STATE(t_a) =$ complete and $SUB_{d(t_c,t_a)} =$ incomplete.

start_next_step($t^{j+1}_{OPP(d_j)}$, $t^{j+1}_{d_{j+1}}$, d_j, d_{j+1}):

1. Let $t_c = t^{j+1}_{OPP(d_j)}$. Set $TRANS(t_c) =$ ready. While $t_c \neq t^{j+1}_{d_{j+1}}$:
   (a) Let t_a denotes the tile adjacent to t_c in direction $KEY_{d_{j+1}}(t_c)$. Set $TRANS(t_a) = TRANS(t_c)$ and clear $TRANS(t_c)$. Then let $t_c = t_a$.

reset($t^m_{OPP(d_{m-1})}$):

1. Set $TRANS(t^m_{OPP(d_{m-1})}) =$ reset. For all tiles t_a adjacent to $t_c = t^m_{OPP(d_{m-1})}$ in directions $d \in NEXT(t_c) \cup PREV(t_c)$, set $TRANS(t_a) = TRANS(t_c)$ and set $t_c = t_a$.
2. If $TERM(t_c)$, set $t_c = RESET(t_c)$. For the tile $t_a \in PREV_{t_c}(t_c)$, set $SUB_{d(t_c,t_a)}(t_a) =$ done. Add $d(t_c, t_a)$ to $NEXT(t_a)$ if not already done.
3. If $length(SUB_{NEWS}(t_c) =$ done $) = length(NEXT(t_c))$, then for the tile $t_a = PREV_{t_c}(t_c)$, set $t_c = RESET(t_c)$ and set $SUB_{d(t_c,t_a)}(t_a) =$ done.

References

[1] R. M. Alaniz, D. Caballero, S. C. Cirlos, T. Gomez, E. Grizzell, A. Rodriguez, R. Schweller, A. Tenorio, and T. Wylie. Building squares with optimal state complexity in restricted active self-assembly. *Journal of Computer and System Sciences*, 138:103462, 2023.

[2] A. A. Cantu, A. Luchsinger, R. Schweller, and T. Wylie. Signal Passing Self-Assembly Simulates Tile Automata. In Y. Cao, S.-W. Cheng, and M. Li, editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[3] C. T. Chalk, D. A. Fernandez, A. Huerta, M. A. Maldonado, R. T. Schweller, and L. Sweet. Strict self-assembly of fractals using multiple hands. 76(1):195–224, sep 2016.

[4] J. Hendricks, M. Olsen, M. J. Patitz, T. A. Rogers, and H. Thomas. Hierarchical self-assembly of fractals with signal-passing tiles. *Natural computing*, 17:47–65, 11 2018.

[5] J. Hendricks and J. Opseth. Self-assembly of 4-sided fractals in the two-handed tile assembly model. In M. J. Patitz and M. Stannett, editors, *Unconventional Computation and Natural Computation*, pages 113–128, Cham, 2017. Springer International Publishing.

[6] J. Hendricks, J. Opseth, M. J. Patitz, and S. M. Summers. Hierarchical growth is necessary and (sometimes) sufficient to self-assemble discrete self-similar fractals. *Natural computing*, 13:357–374, 12 2020.

[7] J. E. Padilla, M. J. Patitz, R. T. Schweller, N. C. Seeman, S. M. Summers, and X. Zhong. Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. *International Journal of Foundations of Computer Science*, 25:459–488, 2014.

[8] M. J. Patitz. An introduction to tile-based self-assembly. In J. Durand-Lose and N. Jonoska, editors, *Unconventional Computation and Natural Computation*, pages 34–62, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] M. J. Patitz and S. M. Summers. Self-assembly of discrete self-similar fractals. *Natural computing*, 9:135–172, 08 2010.

[10] P. W. K. Rothemund. *Theory and experiments in algorithmic self -assembly*. PhD thesis, 2001.

[11] P. W. K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of dna sierpinski triangles. *PLOS Biology*, 2(12):null, 12 2004.

[12] E. Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, 1998.