Self-Assembled Circuit Patterns

Matthew Cook, Paul W.K. Rothemund, and Erik Winfree

Computer Science and Computation & Neural Systems
California Institute of Technology, Pasadena, CA 91125, USA

Abstract. Self-assembly is a process in which basic units aggregate under attractive forces to form larger compound structures. Recent theoretical work has shown that pseudo-crystalline self-assembly can be *algorithmic*, in the sense that complex logic can be programmed into the growth process [26]. This theoretical work builds on the theory of two-dimensional tilings [8], using rigid square tiles called Wang tiles [24] for the basic units of self-assembly, and leads to Turing-universal models such as the Tile Assembly Model [28]. Using the Tile Assembly Model, we show how algorithmic self-assembly can be exploited for fabrication tasks such as constructing the patterns that define certain digital circuits, including demultiplexers, RAM arrays, pseudowavelet transforms, and Hadamard transforms. Since DNA self-assembly appears to be promising for implementing the arbitrary Wang tiles [30, 13] needed for programming in the Tile Assembly Model, algorithmic self-assembly methods such as those presented in this paper may eventually become a viable method of arranging molecular electronic components [18], such as carbon nanotubes [10, 1], into molecular-scale circuits.

1 Introduction

A simple example of embedding computation in self-assembly is shown in Figure 1 (from [29]). The seven square tiles pictured in Figure 1(a) are Wang tiles [24]; they are to be arranged so that labels on the sides of abutting tiles match. Many copies of each tiles may be used, but the tiles may not be flipped or rotated. The result is a pattern such as the one shown in Figure 1(c).

To be applicable to the subject of self-assembly, Wang's tiling model must be extended to describe *how* the tiles aggregate into patterns, based on simple local rules. The Tile Assembly Model [28] does this by assigning an integer *bond strength* to each side of each tile. Growth occurs by the addition of single tiles, one at a time. In order for a new tile to attach itself to an existing pattern of tiles, the sum of the bond strengths on the edges where it would stick must sum to at least the threshold τ, a fixed parameter of the experiment.

The tiles shown in Figure 1(a) constitute a *self-assembly program* for counting in binary, and we will refer to them in this paper as the *counter tiles*. Lines on the edges are drawn to indicate the strength of binding: a thin line indicates a strength-1 bond, thin double lines indicate a strength-2 bond, and a thick line indicates a strength-0 bond (i.e. a side that does not stick to anything). Of course, a bond is formed only when the edge labels match.

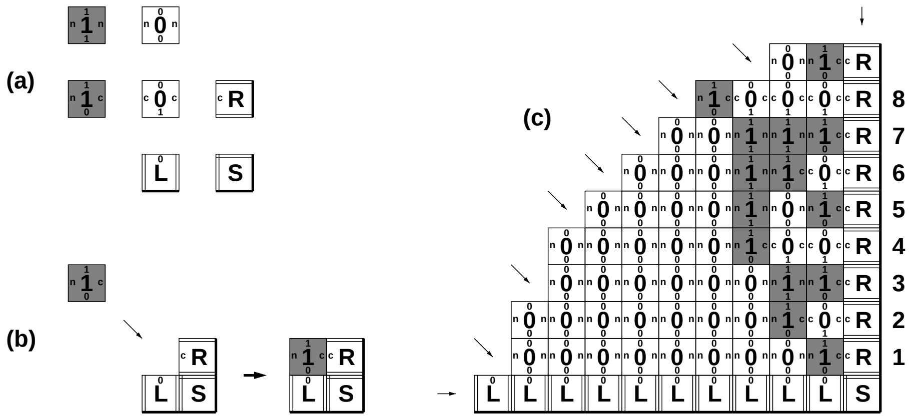

Fig. 1. The counter tiles (from [29]). The set of seven tiles shown in (a) are a Tile Assembly Model program for counting in binary. The tiles labeled “1” are colored gray to make it easier to see the resulting pattern, visible in (c). The self-assembly progresses by individual tiles accreting to the assembly as shown in (b). Edges marked with a small letter or number have bond strengths of 1, while edges with a double line have bond strengths of 2 (and do not require a further label here, since there is only one vertical and one horizontal kind). A later stage of self-assembly is shown in (c), with arrows indicating all the places that a new tile could accrete.

To understand how the program works, we can conceptually categorize the seven tiles used in this example into two groups: The three tiles bearing large letters, called *boundary tiles*, are used to set up the initial conditions on the boundary of the computation. The four tiles bearing large numbers, called *rule tiles*, perform the computation and their numbers are to be interpreted as the binary digits of the output pattern.

The pattern in Figure 1(c) shows a stage of self-assembly with $\tau = 2$, so tiles can only bind to one another when the total binding strength is ≥ 2. For example, an “L” tile may bond on either side to another “L” tile or on its right side to an “S” tile, using a single strength-2 bond. The rule tiles, which can form only strength-1 bonds, can only bind to an assembly if two or more bonds cooperate to hold the tile in place, since $\tau = 2$. Thus, at first, the only counter tiles which can assemble are boundary tiles, via strength-2 bonds. Only after the boundary tiles have begun to assemble into a V-shape, can rule tiles begin binding at *corner sites* as shown in Figure 1(b). The rule tile shown there can form two strength-1 bonds, and it is the only tile that can stick there.

Successive additions of rule tiles and boundary tiles would result in a structure like that in Figure 1(c) whose rows may be read, from bottom to top, as an enumeration of binary numbers. To understand how this works, inspect the rule tiles. Consider the bottom and right sides of each rule tile as *inputs*, and the left and top sides as *outputs*. A rule tile fitting into a corner “reads” two input bits by matching bonds; one bit it reads is the identity of the digit below it and the other is the carry bit from the tile to its right (if “c”, carry= 1; if “n”, carry$= 0$). The number on the rule tile and the bond that it outputs on its top reflect the result of adding, modulo 2, the two input bits; the bond it outputs to its left reflects the resulting carry bit. Rule tiles thus copy the digits below them, unless a carry is indicated from the right. Initially the "L" boundary tiles present all zeros to the rule tiles from below; this starts the counting at zero. The "R" boundary tiles present a new carry bit for each row of the counter from the right; this adds 1 to each successive row of the counter.

It is clear from Figure 1(c) that multiple corner sites may be available for binding rule tiles at the same time; the order in which tiles are added at these sites is not specified. Despite the nondeterministic nature of assembly, it can be shown that the infinite structure that is formed by the counter tiles is unique [27]. This is essentially because a unique rule tile binds at each corner site. For the counter tiles, this in turn is a consequence of our requirement that a rule tile may be added only by the cooperative formation of at least two bonds at once, that is, $\tau = 2$ while the rule tile bond strengths are each 1.

To understand why we use $\tau = 2$, consider what would happen if $\tau = 1$. Since bond strengths are required to be integers, any Tile Assembly Model program with $\tau = 1$ would not be able to require a tile to match the assembly on more than a single side, which makes information processing difficult at best, and if a unique output is required, self-assembly at $\tau = 1$ appears not to be Turing-universal in two dimensions.

If $\tau = 2$ is more powerful than $\tau = 1$, then why don't we try even higher values? The two-fold answer is that (A) there does not seem to be much to gain, since most Tile Assembly Model programs already work well with $\tau = 2$, and (B) the experimental conditions must allow a tile to be able to distinguish between a total bond strength of τ vs. a total bond strength of $\tau - 1$, so experimentally it is good to maximize the ratio between these, which means minimizing τ.

Is the $\tau = 2$ assumption reasonable for physical systems? Real crystal growth does seem to approximate $\tau = 2$ growth with strength-1 bonds. The phenomena of faceting and supersaturation are both consequences of the rarity of steps that violate $\tau = 2$. If a *programmable* experimental system well-modeled by $\tau = 2$ (such as [30] or [19]) can be perfected, then two-dimensional self-assembly can be used to build a binary counter, and in fact, two-dimensional $\tau = 2$ self-assembly is universal [26]. That is, any computer program may be translated into a set of tiles that when self-assembled, simulate the computer program. But the stubbornly practical may still ask: *What is such an embedding of computation in self-assembly good for?*

2 Self-Assembled Circuits

In principle we could use self-assembly wherever we use a conventional computer. In practice we do not expect that computation by self-assembly will be able to compete with the speed of traditional computer architectures on explicitly computational problems. Instead, factors such as the physical nature of the output and the ability to run the same program many times at once in parallel motivate

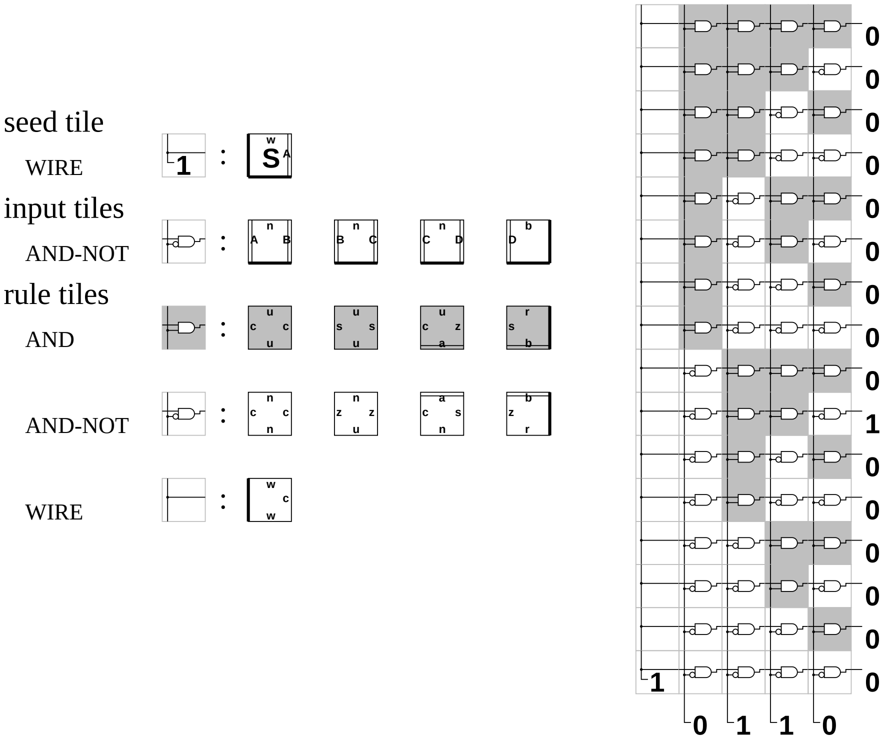

Fig. 2. Using a binary counter to self-assemble a demultiplexer. Logic levels for an example input-output pair are shown: only the row that exactly matches the input pattern is set to "1". To make a pattern with N rows, $10 + \log N$ tiles are used.

us to look for *fabrication problems*: particular patterns or sets of patterns that have potentially useful properties (e.g. as templates for electronic circuits), and which are amenable to self-assembly.

Naively we might wonder, "Can we self-assemble the circuit for a contemporary CPU?" Assuming that we can create tiles that act as circuit elements[1] what we are really asking is "Can we self-assemble the layout pattern for a CPU?" The answer, in theory, is yes, and we may do so without using any complex computation.

Any particular pattern, no matter how complex, can be self-assembled by assigning a unique tile type, with a unique set of binding interactions with its neighbors, to each position in the pattern. The resulting program is as big as the pattern itself, with every tile in the program being used just once in the pattern. This type of self-assembly program (called *unique addressing*) is undesirable because it is not efficient — an efficient program would use a small number of tile

[1] Periodic electrical networks of functional LEDs have already been self-assembled on the millimeter scale [7].

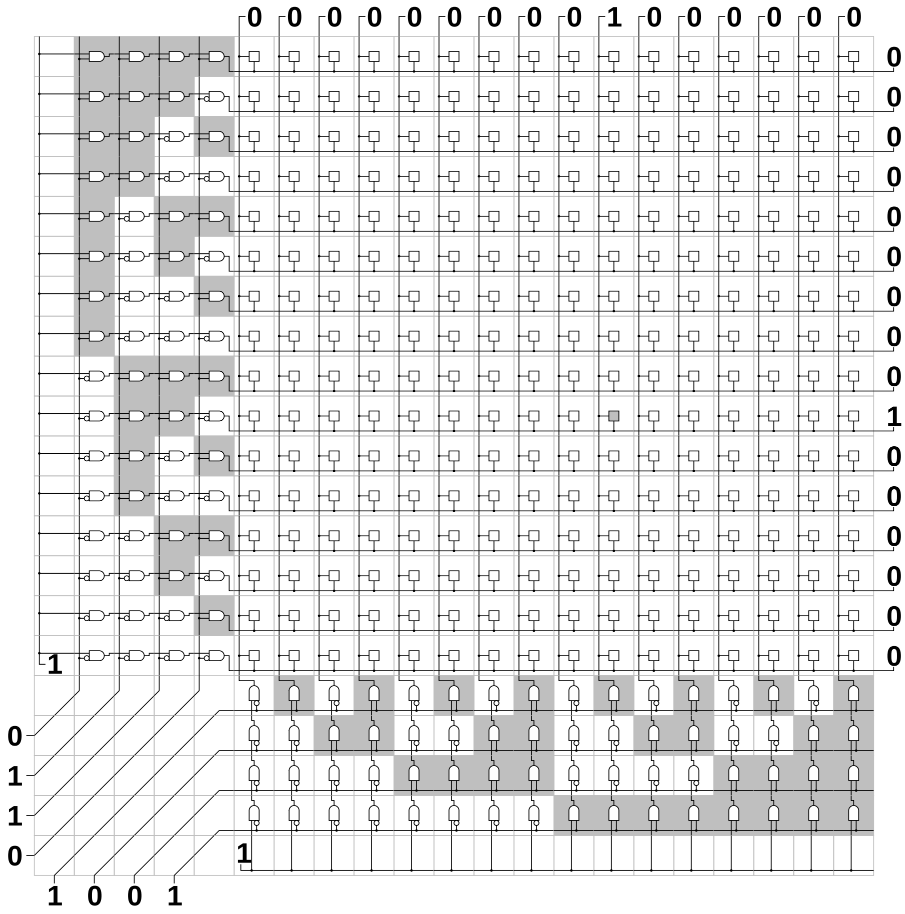

Fig. 3. Two self-assembled demultiplexers at right angles can address a memory. The gray memory cell is being addressed in this figure.

types compared to the size of the pattern. Instead, unique addressing uses the greatest number of tile types possible to create a pattern. In physical implementations [30] it appears that creating unique tile types and unique specific binding interactions is expensive and difficult, so with currently-envisioned techniques it seems that unique addressing is impractical except for very small patterns.

For a circuit to be well-suited to self-assembly, its structure should have a highly methodical pattern to it. The simplest such pattern would be a periodic arrangement of units, such as occurs in a random-access memory circuit, shown in the upper right region of Figure 3. Indeed, using DNA self-assembly to create a molecular-scale memory was suggested in [18]. The pattern generated by the counter tiles of Section 1 is a somewhat more interesting pattern, yet still methodical, which we can see is why it was easy to implement via self-assembly. Later in this paper we will encounter more circuits with methodical structure.

Looking again at the counter tiles, we can think about what similar programs we might be able to construct. The counter tiles use a constant number of tile types to form a structure that grows indefinitely in two directions. If we wish to form a structure of a specific chosen size, we need a set of tiles that not only count, but also stop when the count is complete. Such efficient self-assembly programs for growing finite shapes have been presented in [20]. Here, we use an improved construction [5] wherein, in each successive row, the rightmost "0" is replaced by a "1" and all bits to its right are zeroed. If there is no rightmost "0", it stops. In this construction, shown in Figure 2, a set of $\log N$ *input tiles* are used to define the width of the counter; the assembly grows into a rectangle of exactly size $N \times (1 + \log N)$. Thus the counter can be used to make relatively narrow structures of a chosen length. By adding an additional constant number of tiles we can self-assemble $N \times N$ squares. In these examples, wedding computation with self-assembly addresses what is to chemists a difficult synthetic (fabrication) problem — how to make polymer or crystalline structures of a well-defined size.

Perhaps surprisingly, the binary counter itself happens to yield the layout for a useful circuit. In Figure 2, each tile type is shown labelled with a circuit element, such as a wiring arrangement, an AND gate, or an AND-NOT gate. Once assembled, the tiles form a circuit with 4 input lines along the bottom and $2^4 = 16$ output lines along the right. This is a demultiplexer: the address bits on the input lines specify exactly one output line to be active. A larger circuit with $n = \log N$ input lines and $N = 2^n$ output lines can be self-assembled by changing only the n input tiles. Note that multiple types of tiles can carry the same circuit element. This is a common phenomenon: all the markings on all the tiles comprise rather more information than just the pattern that we care to create; this excess information is necessary to specify how to grow the pattern correctly.

This is our first example of self-assembly being used to create a useful circuit[2]. Whether or not this could be practical depends upon how the tiles are implemented physically and how the circuit elements are attached to the tiles. Let's speculate on a few possible approaches, each of which involves considerable challenges. For example, if the tiles were made of DNA (e.g., the 2×12 nm molecules in [30]) and the circuit elements were small molecular electronic devices (e.g., [6, 14]) covalently attached to the DNA, some chemical post-processing could be necessary to make functional connections between the circuit elements. On the other hand, if again DNA tiles were used but now the labels were single-stranded DNA protruding from the tiles, then in a post-processing step after assembly is complete, larger circuit elements (e.g., DNA-labelled carbon nanotubes [25]) could be arranged by hybridization to the self-assembled pattern, thereby forming the desired circuit. Alternatively, the tiles could be micron- or millimeter- scale objects with embedded conventional electronic components, as

[2] Our approach, in which the self-assembled patterns are used as templates for fabricating functional circuits out of other materials, can be contrasted to work that uses the self-assembly process itself to perform either a fixed [12] or reconfigurable [4] computation.

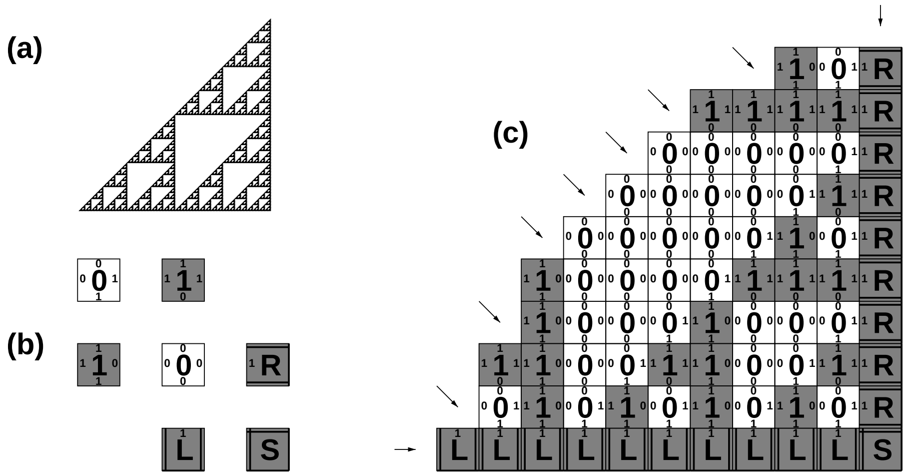

Fig. 4. The Sierpiński triangle and a set of tiles that construct it in the limit.

in [7, 11, 3]. Algorithmic self-assembly has been demonstrated at this scale as well [19].

A demultiplexer could be used as a building block for a larger self-assembled circuit: a pair of demultiplexers oriented at right angles along the borders of an $N \times N$ memory allow a memory element to be accessed using only $2 \log N$ lines. Thus a memory circuit may be self-assembled (see Figure 3). What other circuits might be possible? Our next constructions derive from the observation that the demultiplexer circuit implements a generalized inner product of a binary vector by a binary matrix, with the binary function EQUALS substituting for multiplication and AND substituting for addition in the definition of matrix multiplication. That is, the circuit takes an n-bit binary vector, "multiplies" it by a $n \times 2^n$ size binary "counting" matrix, and outputs a 2^n long vector. Similarly, a circuit for an arbitrary binary matrix multiplication could be created by self-assembling a circuit decorated with logic gates as appropriate for the matrix of choice.

3 Self-similar Transforms

Another complex pattern that may be created by a simple self-assembling computation is the Sierpiński triangle, pictured in Figure 4(a). Only seven tiles, shown in Figure 4(b) (from [27]), are required to create a pattern (shown in Figure 4(c)) whose limit is this triangular fractal pattern. As with the counter tiles, its construction depends on $\tau = 2$ assembly. By labeling the sides of the tiles as "input" and "output", individual tiles can be seen to encode the binary function XOR. Diagonals of the assembly, interpreted as zeros and ones, form rows of Pascal's triangle modulo 2. It can also be seen that diagonals of the assembly are instantaneous descriptions of a one-dimensional cellular automaton. Aside

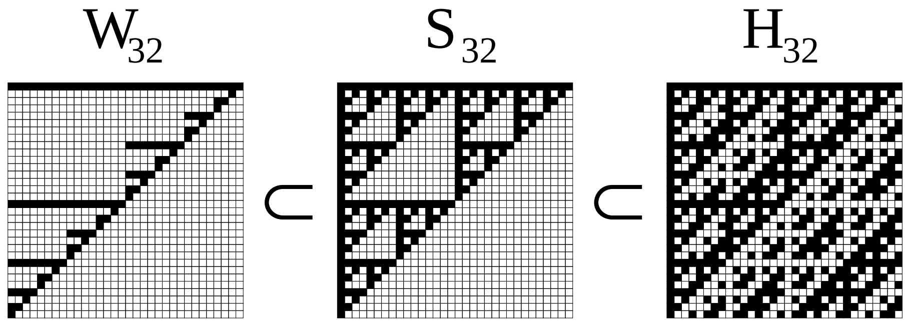

Fig. 5. Comparison of self-similar binary matrices. From left to right, the binary pseudowavelet matrix (W_{32}), the Sierpiński triangle matrix (S_{32}), and the Hadamard matrix (H_{32}). For the pseudowavelet and Sierpiński matrices, black represents 1 and white represents 0. For the Hadamard matrix, black represents 1 and white represents -1.

from its interpretation as a computation, this pattern is beginning to find some practical uses; rendered in metal the Sierpiński triangle appears to be a superior cellular phone antenna [17].

Does the Sierpiński triangle also have a circuit interpretation like the binary counter? Perhaps not, but it inspires thought: interpreted as a binary matrix the Sierpiński triangle has many periodic rows whose periods are related by a logarithmic scaling. This suggests that using the Sierpiński triangle as a matrix multiplier might effect some transform similar to a wavelet or Fourier transform. In fact, binary versions of the wavelet and Fourier transforms, namely the binary pseudowavelet transform [15] and the Hadamard transform [21, 16, 31], have self-similar matrices closely related to the Sierpiński triangle. Both these transforms have been used in signal processing and image compression. The Hadamard matrix in particular has uses from quantum computation to cell phones, and can be used directly for implementing a parallel Walsh transform [23]. Many theoretical and practical uses have been studied for Hadamard matrices of size 2^n [2, 9, 22].

Given the similarity of these transforms to the Sierpiński triangle, it seems reasonable to expect that there should exist simple tile sets that self-assemble into circuit patterns for computing them. This turns out to be correct.

In Figure 5 we give a visual comparison of these matrices to the Sierpiński matrix. Their formal similarity can be seen from their recursive definitions: $W_1 = T_1 = H_1 = S_1 = [1]$, and for n a power of 2,

$$W_{2n} = \begin{bmatrix} T_n & W_n \\ W_n & 0 \end{bmatrix}, \quad T_{2n} = \begin{bmatrix} T_n & T_n \\ 0 & 0 \end{bmatrix}, \quad S_{2n} = \begin{bmatrix} S_n & S_n \\ S_n & 0 \end{bmatrix}, \quad H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix}$$

The pseudowavelet transform W_n has a simple self-similar structure for which it seems likely we can find a simple self-assembly program; in fact, a straightforward modification of the Sierpiński tiles will suffice. First, modify the tiles so growth occurs from the right to the left (in Figure 5); then make a “tagged”

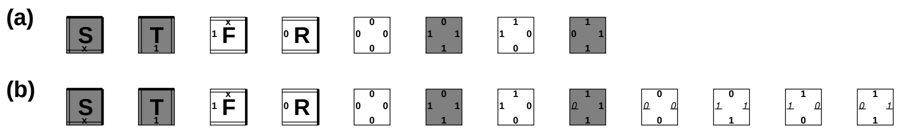

Fig. 6. Construction of the pseudowavelet tile set. (a) A tile set for growing the Sierpiński triangle from the upper right corner, as in Figure 5. (b) A tile set for growing the pseudowavelet transform from the upper right corner.

Fig. 7. The two types of hexagonal tile that will be used for constructing the pattern in Figure 8(c).

version of each rule tile such that in each row, the first "0" to the left of a "1" gets tagged, and tags propogate leftward. The black cells are defined by only untagged tiles. These tiles are shown in Figure 6. Although these tiles build unbounded patterns, patterns of defined size can be created by replacing the 4 boundary tiles with a binary counter, as in Figure 2 and Figure 3.

4 Growing a Hadamard matrix

In this section we will present a set of hexagonal tiles which deterministically constructs self-similar Hadamard matrices of order 2^n. We begin, in Subsection 4.1, by presenting a simple set of "red and green" tiles which constructs a nice but non-Hadamard self-similar pattern. Then, in Subsection 4.2 we will present a slight elaboration on those tiles which results in the generation of a Hadamard matrix and indicate how to turn the given construction into one that works with square tiles. Finally, in Subsection 4.3 we prove the correctness of our construction.

4.1 Red and Green Tiles

Figure 7 shows two hexagonal types of tiles, one red and one green. Unlike the square Wang tiles discussed in earlier sections, these tiles may be rotated and/or flipped over. Where two tiles abut, the notches on the sides of the hexagons must fit together (one out and one in), or, where there are spots instead of notches, the spots of the tiles must match. During growth, a new hexagon will need to fit in with three existing hexagons, so we will have $\tau = 3$. (Later we will show how τ can be reduced to 2 by converting the hexagons into square tiles.)

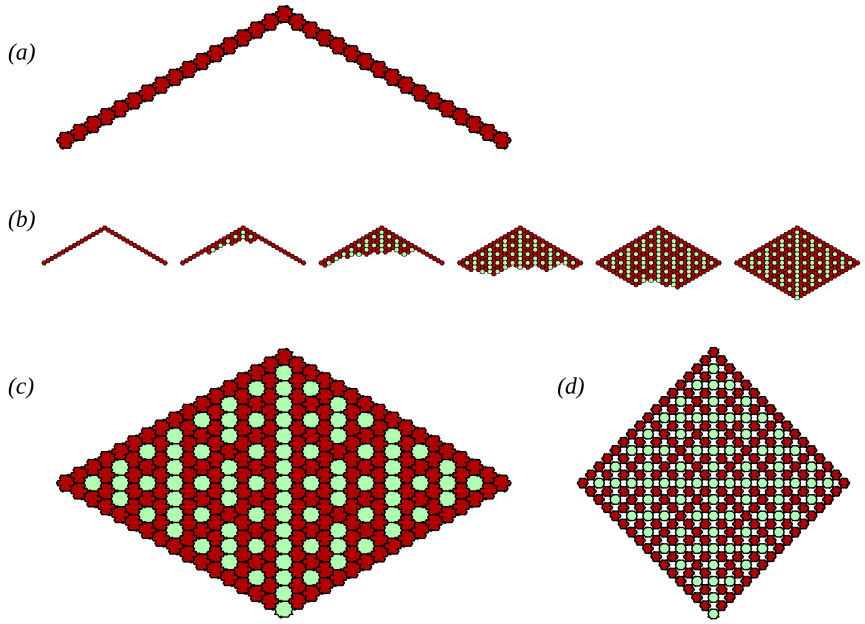

Fig. 8. Stages of growth for the tiles from Figure 7. (a) shows the boundary condition used to start the growth. (b) shows a sample sequence of snapshots as it grows over time. (c) shows the pattern after it has fully filled in. (d) shows exactly the same pattern as (c), but with the tiles pulled apart vertically so that the overall shape is now a square. The relationship to Figure 9 begins to become apparent.

The boundary condition used to initiate growth is composed of red tiles as shown in Figure 8(a). As in the construction shown in Figure 1, three boundary tile types with strength-3 bonds can be used to construct this initial condition, or tiles analogous to those in Figure 2 could be used to self-assemble a boundary of size exactly 2^n.

As the assembly grows, as shown in Figure 8(b), the resulting pattern is unique, since at any location where we might try to add a tile, the two notches and the spot always restrict our options so that there is at most one way to add a tile. We can see this by examining just 3 cases: If both notches point up, then we must add a green tile oriented according to the spot. If both notches point down, we must add a red tile (upside down from the one shown in Figure 7) oriented according to the spot. If one notch is down and one is up, then we must use a red tile, but it will only fit if the spot is on the side where the notch points up. Luckily, we can prove that the spot will indeed always be on this side, as we will show in Section 4.3.

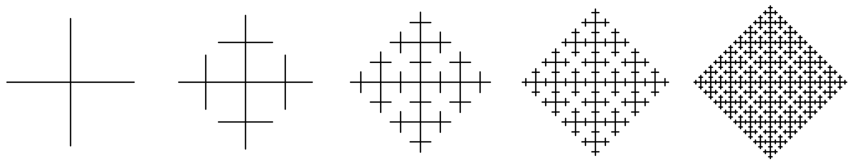

Fig. 9. Successive stages of the “Plus” fractal underlying the pattern of red and green tiles in Figure 8(c,d).

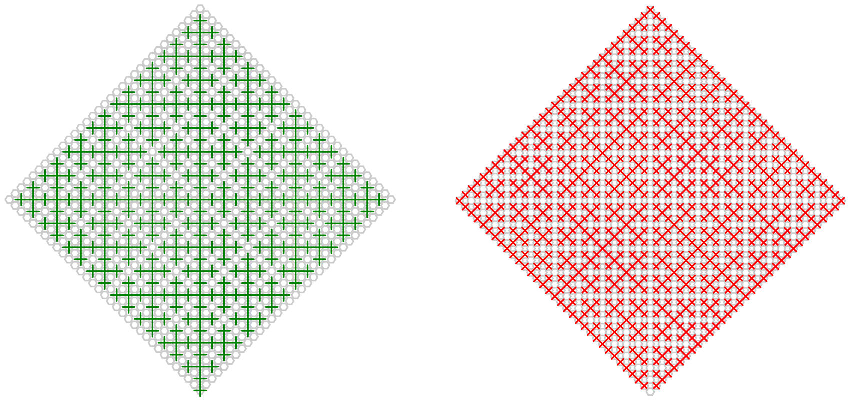

Fig. 10. The green tiles of the pattern in Figure 8(d) happen to be arranged exactly like the “Plus” fractal of Figure 9. Red tiles are also arranged the same way, except the fractal is rotated 45° and centered on an empty point.

Here we will make some observations about the pattern produced by the tiles, without worrying about proofs. Then in Section 4.3 we will prove that the grown pattern has the self-similar nature being discussed.

Given a fully grown pattern of size $2^n + 1$, we can vertically stretch the rhomboidal array of hexagons from Figure 8(c) into a square array as shown in Figure 8(d) to make the self-similar pattern more apparent. The pattern is the self similar pattern of the “Plus” fractal shown in Figure 9. As shown in Figure 10, if we draw a green + on every green tile in the pattern, then we see exactly the “Plus” fractal, while if we draw a red X on every red tile, we see a simple rearrangement of the same fractal.

In fact, we can draw both the red and the green pattern together on the same tiling, as in Figure 11(a), and in spite of them each covering the entire figure, the red and green fractals do not touch each other at all.

Since these “Plus” fractals have dimension 2, and adjacent tiles in the red pattern are $\sqrt{2}$ times closer than adjacent tiles in the green pattern, there are twice as many red tiles as there are green tiles.

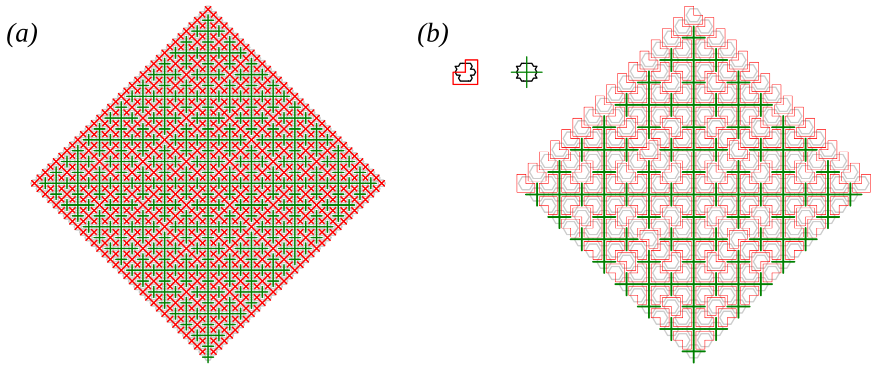

Fig. 11. (a) shows the red and green patterns together. (b) shows the correspondence between the "Plus" fractal and the recursive "L triomino" tiling.

On each red tile, instead of drawing a red X, we can draw an L triomino oriented according to the tile, yielding the well-known recursive tiling shown in Figure 11(b).

4.2 The Hadamard Tiles

Now we will modify the red and green tiles to get a set of tiles that can generate a Hadamard matrix. The main modification is just that we will add +1 and −1 markings to the tiles, so we will have a +1 red tile, a −1 red tile, a +1 green tile, and a −1 green tile. The +1 and −1 markings on these tiles are what will form the Hadamard matrix pattern.

We will have the red boundary tiles (corresponding to Figure 8(a)) all carry the +1 marking. The information about whether a tile is marked +1 or −1 will be propagated similarly to how the 0 and 1 markings were propagated in the tiles for the Sierpiński triangle, by labeling edges with "input" and "output" values. Specifically, on each tile we will label the two lower notched edges (the "output" edges) with the tile's main marking, while the two upper notched edges (the "input" edges) will be labeled with compatible inputs. Note that this means we can no longer rotate or flip our tiles, so we will need to explicitly have both orientations of the green tile, and all four orientations of the red tile.

On a red tile, the inputs may be either the same or different, and the tile's main marking always matches the input on the same side as the spot. On a green tile, the inputs are always the same, and the tile's main marking is always the opposite of the inputs. This results in a total of 16 types of red tile and 4 types of green tile.

Note that if we wanted to use square tiles instead of hexagonal tiles, we could eliminate the sides with the spots, and instead communicate the handedness of the spot via the tile to the left of what are now two square tiles touching only at a corner. This breaks some of the symmetry of the tile set (although

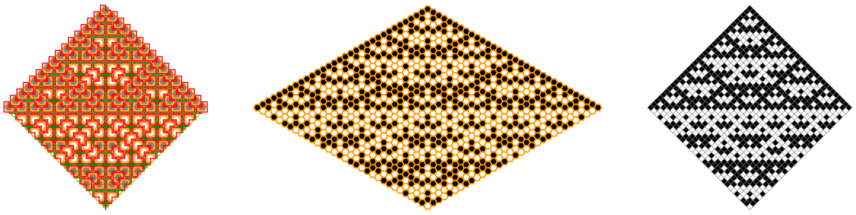

Fig. 12. Three depictions of the self-assembled Hadamard matrix. The leftmost diagram shows how the green/red form of the "Plus"/"L triomino" pattern is related to the Hadamard pattern: Every green tile has the opposite Hadamard color from the tiles above it, while every red tile has the same color as the tile above it in the direction of its axis of symmetry. This surprising relationship between these two fundamental self-similar patterns is the key to how the easily-constructed red and green pattern is used as a stepping stone to the more difficult Hadamard pattern.

the marking could also be redundantly communicated on the right as well, to preserve symmetry), but if one needs to use $\tau = 2$ square tiles, it is nice to know that there is no theoretical obstacle.

Figure 12 shows the Hadamard pattern as it is produced on the hexagons together with the red and green patterns, on its own in the original hexagonal setting, and in its square matrix form.

Now we know what to expect when we grow our pattern. Of course, to get a pattern of size exactly 2^n for some given n, one would need to start with a boundary condition of just the right length, which can be accomplished as described in Section 2.

4.3 Proof of Correctness

In this section we prove that the iterative process of tile accretion generates exactly the same pattern as the recursive subdivision process shown in Figure 13.

Since we know from Section 4.1 that at any given position there is only one way a tile can be added to that location, we know that the pattern that grows is the unique pattern satisfying the boundary condition and the edge matching conditions on the tiles. (If there were more than one growable pattern, then the uppermost tile position differing in two such patterns would indicate a place where there is more than one way to add a tile.)

This means that if we can show that the recursive subdivision process always yields an arrangement that is consistent with the growth rules for the notch and spot markings on the edges as well as the Hadamard markings, then it must yield exactly the same arrangement as is generated by the tile accretion process.

To show that the recursive subdivision process never leads to an inconsistency among the tiles, we consider what happens when we subdivide every tile in a consistent pattern X to get a more detailed pattern Y. We will show that if X was consistent, then so is Y.

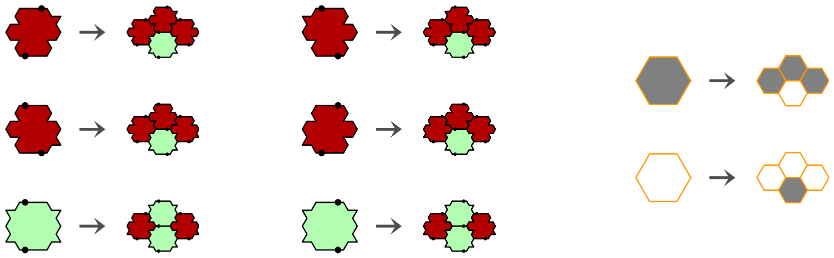

Fig. 13. The left two columns show how red and green hexagons get subdivided into four hexagons. The second column is the mirror image of the first. Every edge of the four small hexagons either is the same in all 6 subdivisions, or takes its notch or spot from a fixed edge of the parent hexagon for all 6 subdivisions. The third column shows how the Hadamard markings are placed on the subdivided hexagons: shaded hexagons represent $+1$, and white hexagons represent -1. The Hadamard markings coexist with the notch and spot markings, but are shown separately for clarity. If they were shown together, the first two columns would each need to be shown twice: once with the upper Hadamard shadings of column 3, and once with the lower Hadamard shadings of column 3.

First we consider the spots. The accretion rule for spots is that the spots must line up on vertically adjacent tiles. Does the subdivision process guarantee that this will be the case throughout any pattern Y which is the result of subdividing a consistent pattern X? There are three cases where tiles in Y need to agree on the spot position: Two vertically adjacent tiles in Y may have come from (A) the same tile in X, (B) vertically adjacent tiles in X, or (C) diagonally adjacent tiles in X. For case (A), we know the spots will agree because we see that they agree in the interior of each individual subdivision rule. For case (B), we know the spots will agree because we see that the alignment of the spots at the top and bottom of each subdivision quadruple match the alignment of the spots at the top and bottom of the parent hexagon, and we know the parent hexagons agreed in X. For case (C), we know the spots will agree because both the lower and upper quadruple have the spot towards the top tile of the lower quadruple, regardless of what quadruples were used.

Next we consider the notches. The accretion rule for notches is that they must match in direction on diagonally adjacent tiles. The subdivision process leads to three cases where two tiles in Y need to agree on the notch direction: (A) the two tiles are in the same quadruple, (B) the two tiles are the top tile of the lower quadruple and a side tile of the upper quadruple, or (C) the two tiles are a side tile of the lower quadruple and the bottom tile of the upper quadruple. For case (A), we know the notches will match because we can see that they match in every possible quadruple. For case (B), we know the notches will agree because regardless of which quadruples are involved, the notch in question will always match the original notch connecting the two parent tiles in X. For case (C), we know the notches will agree because regardless of which quadruples are involved, the notch will always point down.

Finally we consider the Hadamard markings. The accretion rule for the Hadamard markings is that for a red tile, the marking is copied from the upper left or upper right tile on the side of the spot, while for a green tile, the marking is the opposite of the markings on the upper left and upper right tiles (which happen to have the same markings). We can immediately see that the the Hadamard markings obtained by subdivision obey the accretion rule for the side and bottom tiles of each quadruple, while for the top tile we need to know something about the upper left and upper right neighbor quadruples. What we know about these quadruples is that their side tiles have the same Hadamard marking as their parent in the X tiling. The top tile of the lower quadruple, whose Hadamard marking we are trying to verify, is also marked the same as *its* parent in the X tiling, and in fact it is always the very same tile as its parent. This means that the correct Hadamard marking for its parent in the X tiling is the same as its correct Hadamard marking in the Y tiling, and so since its parent was indeed marked correctly in the X tiling, we know it will be marked correctly in the Y tiling.

Since the spots, notches, and Hadamard markings present after subdivision follow all the rules used for accretion, we see that the subdivision process does indeed yield the growable patterns. If we start with the first tile shown in the subdivision rules, and repeatedly subdivide it to get patterns with more and more tiles, we see that the upper two sides of the resulting array of hexagons match exactly the boundary condition shown in Figure 8. This means that the pattern obtained by repeated subdivision of this tile is exactly the same pattern that grows from that boundary condition. In particular, the Hadamard markings will be exactly those that occur on the pattern grown from the boundary condition. Since it follows from the definition of Hadamard matrices that they are exactly what gets produced by the Hadamard marking subdivision rule shown in the third column of Figure 13, this means that the Hadamard markings on the grown pattern will exactly match the intended Hadamard matrix.

Acknowledgements

M.C. is supported in part by the "Alpha Project" that is funded by a grant from the National Human Genome Research Institute (Grant No. P50 HG02370). P.W.K.R. is supported by a Beckman Postdoctoral Fellowship. E.W. is supported by NSF Career Grant No. 0093486, DARPA BIOCOMP Contract F30602-01-2-0561, and NASA NRA2-37143. Email may be addressed to `cook@paradise.caltech.edu`.

References

1. A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker. Logic circuits with carbon nanotube transistors. *Science*, 294:1317–1320, November 9 2001.

2. K. G. Beauchamp. *Walsh Functions and Their Applications*. Academic Press, London, 1975.
3. M. Boncheva, D. H. Gracias, H. O. Jacobs, and G. M. Whitesides. Biomimetic self-assembly of a functional asymmetrical electronic device. *PNAS*, 99(8):4937–4940, April 16 2002.
4. A. Carbone and N. C. Seeman. Circuits and programmable self-assembling DNA structures. *PNAS*, 99(20):12577–12582, October 1 2002.
5. Q. Cheng and P. M. de Espanes. Resolving two open problems in the self-assembly of squares. Usc computer science technical report #03-793, University of Southern California, 2003.
6. C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams, and J. R. Heath. Electronically configurable molecular-based logic gates. *Science*, 285:391–394, 1999.
7. D. H. Gracias, J. Tien, T. L. Breen, C. Hsu, and G. M. Whitesides. Forming electrical networks in three dimensions by self-assembly. *Science*, 289:1170–1172, Aug. 18, 2000.
8. B. Grünbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman and Company, New York, 1987.
9. H. F. Harmuth. Applications of walsh functions in communications. *IEEE Spectrum*, 6:82–91, 1969.
10. Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K.-H. Kim, and C. M. Lieber. Logic gates and computation from assembled nanowire building blocks. *Science*, 294:1313–1317, November 9 2001.
11. H. O. Jacobs, A. R. Tao, A. Schwartz, D. H. Gracias, and G. M. Whitesides. Fabrication of a cylindrical display by patterned assembly. *Science*, 296:323–325, April 12 2000.
12. M. G. Lagoudakis and T. H. LaBean. 2D DNA self-assembly for satisfiability. In E. Winfree and D. K. Gifford, editors, *DNA Based Computers V*, volume 54 of *DIMACS*, pages 141–154. American Mathematical Society, Providence, Rhode Island, 2000.
13. C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407(6803):493–496, 2000.
14. A. R. Pease, J. O. Jeppesen, J. F. Stoddart, Y. Luo, C. P. Collier, and J. R. Heath. Switching devices based on interlocked molecules. *Acc. Chem. Res.*, 34:433–444, 2001.
15. S. Pigeon and Y. Bengio. Binary pseudowavelets and applications to bilevel image processing. *Data Compression Conference (DCC '99)*, pages 364–373, 1999.
16. W. Pratt, J. Kane, and H. Andrews. Hadamard transform image coding. *Proceedings of the IEEE*, 57(1):58–68, 1969.
17. C. Puente-Baliarda, J. Romeu, R. Pous, and A. Cardama. On the behavior of the sierpinski multiband fractal antenna. *IEEE Transactions on Antennas and Propagation*, 46(4):517–524, 1998.
18. B. H. Robinson and N. C. Seeman. The design of a biochip: A self-assembling molecular-scale memory device. *Protein Engineering*, 1(4):295–300, 1987.
19. P. W. K. Rothemund. Using lateral capillary forces to compute by self-assembly. *PNAS*, 97:984–989, 2000.
20. P. W. K. Rothemund and E. Winfree. The program size complexity of self-assembled squares. *Symposium on the Theory of Computing (STOC 2000)*, 2000.

21. J. J. Sylvester. Thoughts on orthogonal matrices, simultaneous sign-successions, and tessellated pavements in two or more colours, with applications to newton's rule, ornamental tile-work, and the theory of numbers. *Phil. Mag.*, 34:461–475, 1867.
22. S. G. Tzafestas. *Walsh Functions in Signal and Systems Analysis and Design*. Van Nostrand Reinhold, New York, 1985.
23. J. L. Walsh. A closed set of normal orthogonal functions. *Amer. J. Math*, 45:5–24, 1923.
24. H. Wang. Proving theorems by pattern recognition. II. *Bell System Technical Journal*, 40:1–42, 1961.
25. K. A. Williams, P. T. Veenhuizen, B. G. de la Torre, R. Eritja, and C. Dekker. Carbon nanotubes with DNA recognition. *Nature*, 420:761, 2002.
26. E. Winfree. On the computational power of DNA annealing and ligation. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers*, volume 27 of *DIMACS*, pages 199–221, Providence, Rhode Island, 1996. American Mathematical Society.
27. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, Computation and Neural Systems Option, 1998.
28. E. Winfree. Simulations of computing by self-assembly. Technical Report CS-TR:1998.22, Caltech, 1998.
29. E. Winfree. Algorithmic self-assembly of DNA: Theoretical motivations and 2D assembly experiments. *Journal of Biomolecular Structure & Dynamics*, pages 263–270, 2000. Special issue S2.
30. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.
31. R. Yarlagadda and J. Hershey. *Hadamard Matrix Analysis and Synthesis With Applications to communications and Signal/Image Processing*. Kluwer Academic Publishers, Boston, 1997.