

How Crystals that Sense and Respond to Their Environments Could Evolve

Rebecca Schulman and Erik Winfree

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

Abstract. An enduring mystery in biology is how a physical entity simple enough to have arisen spontaneously could have evolved into the complex life seen on Earth today. Cairns-Smith has proposed that life might have originated in clays which stored genomes consisting of an arrangement of crystal monomers that was replicated during growth. While a clay genome is simple enough to have conceivably arisen spontaneously, it is not obvious how it might have produced more complex forms as a result of evolution. Here, we examine this possibility in the tile assembly model, a generalized model of crystal growth that has been used to study the self-assembly of DNA tiles. We describe hypothetical crystals for which evolution of complex crystal sequences is driven by the scarceness of resources required for growth. We show how, under certain circumstances, crystal growth that performs computation can predict which resources are abundant. In such cases, crystals executing programs that make these predictions most accurately will grow fastest. Since crystals can perform universal computation, the complexity of computation that can be used to optimize growth is unbounded. To the extent that lessons derived from the tile assembly model might be applicable to mineral crystals, our results suggest that resource scarcity could conceivably have provided the evolutionary pressures necessary to produce complex clay genomes that sense and respond to changes in their environment.

1 Introduction

Developments in DNA computing have shown that computation can be embedded in crystal growth processes [31, 23], with potential applications to combinatorial search problems [17, 19] and to fabrication tasks in nanotechnology [11, 1]. But does crystal computation have any relevance to what we observe in nature? We speculate here about a possible connection to the origin of life.

The background for this argument is a hypothesis, proposed and developed by Graham Cairns-Smith [5], that the first primitive “organisms” were clay crystals. In this theory, information (the first “genes”) consisted of patterns stored as variations in crystal structure that could be propagated during crystal growth. For example, in some layered silicate clays, there are two distinct layer types that appear in a cross-section as a sequence that could be considered the crystal’s genotype. Replication occurs by periodic physically-induced fragmentation of crystals into smaller pieces containing the same genotype, leading to exponential increase in the number of organisms. Cairns-Smith considered several types of selective pressures that could have been present and would have resulted in favoring the growth of crystals with non-trivial genotypes. For example, the structure of a layer sequence could result in different crystal morphologies and different susceptibilities to fragmentation. He further envisioned the clays interacting with organic molecules, somehow leading to novel clay-produced organic molecules and eventually to the genetic takeover of organic life forms [6].

One of the strengths of Cairns-Smith’s hypothesis is that it is easy to see how Darwinian evolution could have gotten started by geological processes. However, while clays are known to interact with organic molecules [20, 15], it is hard to know whether these interactions could have provided evolutionary pressure toward increasing complexity. It is therefore interesting to ask whether there are other possible mechanisms that could have stimulated the original evolution of complex sequence information. In fact, it is hard to envision how anything so simple that it could have arisen spontaneously could have evolved into the remarkable complexity of form and function found in modern biological organisms. The capacity of evolutionary systems to create increasingly complex forms with increasingly adapted function – so-called open-ended evolution – remains

poorly understood. To our knowledge, there are no examples in artificial life [2], *in vitro* chemical evolution [36, 16], or *in vivo* directed evolution [37] that have convincingly demonstrated open-ended evolution.

The conceptual and physical simplicity of crystal evolution makes it a promising place to examine these issues. In this paper, we consider whether there are properties of crystal growth that can lead to open-ended evolution. Examining the capacity for interesting evolutionary landscapes requires distinguishing between crystal genotype and phenotype – what is the genetically-determined function performed by crystals that gives rise to a selective advantage? In this paper, we investigate the ability of crystals to respond to selection pressures using computations that occur during growth.

The notion that a crystal can perform a computation is based on the observation that the tiling problem (the question of whether a set of geometric shapes can tile the plane) is undecidable: any problem solvable by a Turing machine can be expressed as a question of whether a particular set of shapes can tile the plane [29]. This observation led to a constructive method of computing by arranging tiles into a lattice [31]. These tiles can be viewed as analogues for crystal monomers; attachment at a specific site in a growing crystal is determined by how well a tile’s shape fits in the growth site. The binding of a particular tile at a particular site can be viewed as a computational or information transfer step.

It is reasonable to assume that crystals grow in environments where their monomers are present in solution at different (possibly time-varying) concentrations. In such an environment, which crystal patterns grow the fastest? Are there environments in which crystals must perform computations in order to grow quickly? We show that crystals can sense the environment and respond by making use of the most abundant tiles. We call this feature a “crystal metabolism” because the computation they perform controls the extraction of resources from the environment. In particular, we are interested in whether there are environments that lead to the evolution of increasingly complex crystal metabolisms.

We examine these issues in principle within a previously described tile assembly model [24], which is a generalized crystal growth model that has been used to study algorithmic self-assembly of synthetic DNA tiles [35, 23, 1]. It has been previously argued [25] that DNA tile crystals have the capacity for Darwinian evolution of the sort imagined by Cairns-Smith. Insights derived from studying this model should be applicable to crystals composed of a variety of other materials, such as proteins [14, 10], RNA [9], or even macroscopic tiles [4, 22] or clay minerals [7].

The tile assembly model describes crystal monomers as square or rectangular tiles with each unit edge labeled to indicate how it fits with other monomers. Crystal growth proceeds by accretion, with single tiles being added to the crystal at sites where they make a sufficient number of contacts. In this paper we consider a version of this model in which (a) a tile may be added to a site if labels on at least two edges match those presented by the crystal at that site, and (b) monomer tiles arrive at potential binding sites with a frequency proportional to their concentration in solution. Occasional violations of rule (a) are referred to as “mutations”. A system consists of a set of tiles, along with the concentrations of those tiles in solution. Because of the particular choice of matching rules, each tile set implicitly determines what arrangements of tiles can grow as crystals. If certain arrangements grow faster than others under given conditions, then we consider these arrangements more fit. Later we will discuss how growth rates relate to the rate of exponential reproduction.

To discuss evolution, it is helpful to consider three aspects of an evolutionary process. First, a self-replicating entity carries information which directs behavior. The space of achievable behaviors is therefore inherently dependent on the material from which they are constructed. Second, there must be an environment in which certain behaviors have selective advantage; this provides the stimulus for evolution to discover complex solutions. Third, for evolution to proceed quickly there must be a route via mutations in which ever more complex behavior is achieved by a series of incremental steps. In crystal evolution, the first aspect (potential for complexity) derives from a choice of a particular tile set; this determines the behavior implicit in the crystal growth. The second aspect (stimulus for complexity) derives from the conditions in which the crystals are grown, e.g. tile concentrations and temperature, etc. The third aspect (the route to complexity) is difficult to predict. We will not address it here; for our purposes, it is enough to know that the fittest crystals could arise through (possibly extremely unlikely) mutations or spontaneous generation. However, understanding the route to complexity is an important goal for future studies of crystal evolution.

In addition to the above, open-ended evolution seems to require that the selective pressure provided by the environment is distinct from the potential for evolution, in the sense that different environments must lead to distinct functional solutions. For example, in modern biology, the universal potential of biochemistry – the ability of DNA to code for proteins and other macromolecules that create seemingly arbitrarily sophisticated and complex machines – makes it possible for living organisms to adapt to an incredible variety of environmental niches, opening the way to ever-increasing complexity. Thus, searching for tile sets capable of open-ended evolution, we first see that a tile set plays the role of a “chemistry” in the sense that it defines the rules by which tile-based “organisms” can grow and function, and we further expect that we are looking for a tile set that displays some sort of universality of behavior.

Following this intuition, we argue that tile-based crystals can exploit computation to enhance their growth rate, and that this can lead to evolutionary processes resulting in increasing complexity of crystal structure. We introduce a framework for studying the evolution of metabolic control in crystals under resource-limited growth conditions. Within this framework, we design tile sets and environments that give rise to evolutionary landscapes in which crystals that perform more effective computations are fitter. We provide two main examples. First, in order to provide simple examples of metabolic evolution, we describe a tile set that can encode programmable logical computations. Second, to emphasize that arbitrarily complex computations can in principle be used by crystals to sense and respond to their environment, we exhibit a tile set capable of universal computation.

2 Zig-Zag Ribbon Evolution

Previously, it was suggested that DNA tile assemblies could in principle evolve through cycles of crystal growth and splitting [25] (Figure 1). The zig-zag tile set described in that work produces ribbon-like crystals that copy information along the length of the crystal. The tile set includes a group of square tiles, and two rectangular tiles called *double tiles*. Logical representations of the tiles that comprise a basic zig-zag ribbon are shown in Figure 2(a).

When growth occurs according to the tile assembly model, tiles are added to a ribbon in a zig-zag pattern shown in Figure 2(b). Only tiles that match at least two edges can attach. Given this constraint, the design of the tiles is such that at any moment there is just one tile that may be added to each end of the ribbon. The addition of each new row can be viewed as the copying of the information in the previous row. Using the tile set shown in Figure 2(a), this copying is trivial – only one sequence type is possible. However, the requirement that a tile attach by two bonds means that it must match both its vertical neighbor, either above and below, and its horizontal neighbor, to the left or right. In the case that several tile types may match the vertical neighbor, as shown in Figure 2(c), only the tile type that already appears in the row will match the horizontal neighbor. Only this tile can be added, so that information is inherited from layer

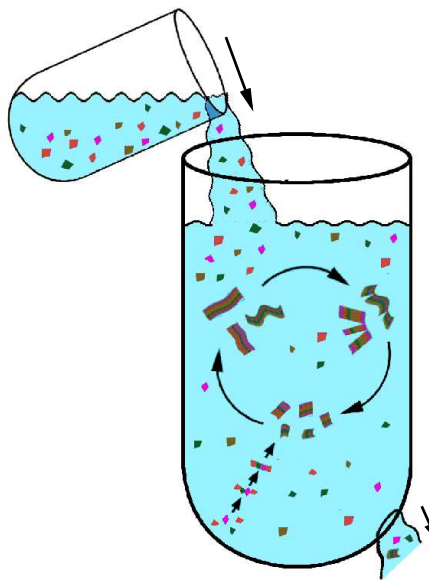


Fig. 1. The zig-zag crystal life cycle. Zig-zag crystals grow by copying their sequences of DNA tiles. Reproduction occurs when a crystal is broken by external mechanical force (e.g. shearing, as shown in the upper right of the test tube). The small crystals that are the result of division (center) continue growing, and eventually become large enough (top left) to split again. The materials required for growth (tiles) are constantly replenished by an inward flow, while an outward flow removes slow-growing crystals from the population. Occasional mutations are propagated during growth and are eventually replicated. Similarly, new assemblies are occasionally generated spontaneously (lower left) from single tiles. Once they reach a certain size, these spontaneously generated assemblies can also grow and reproduce.

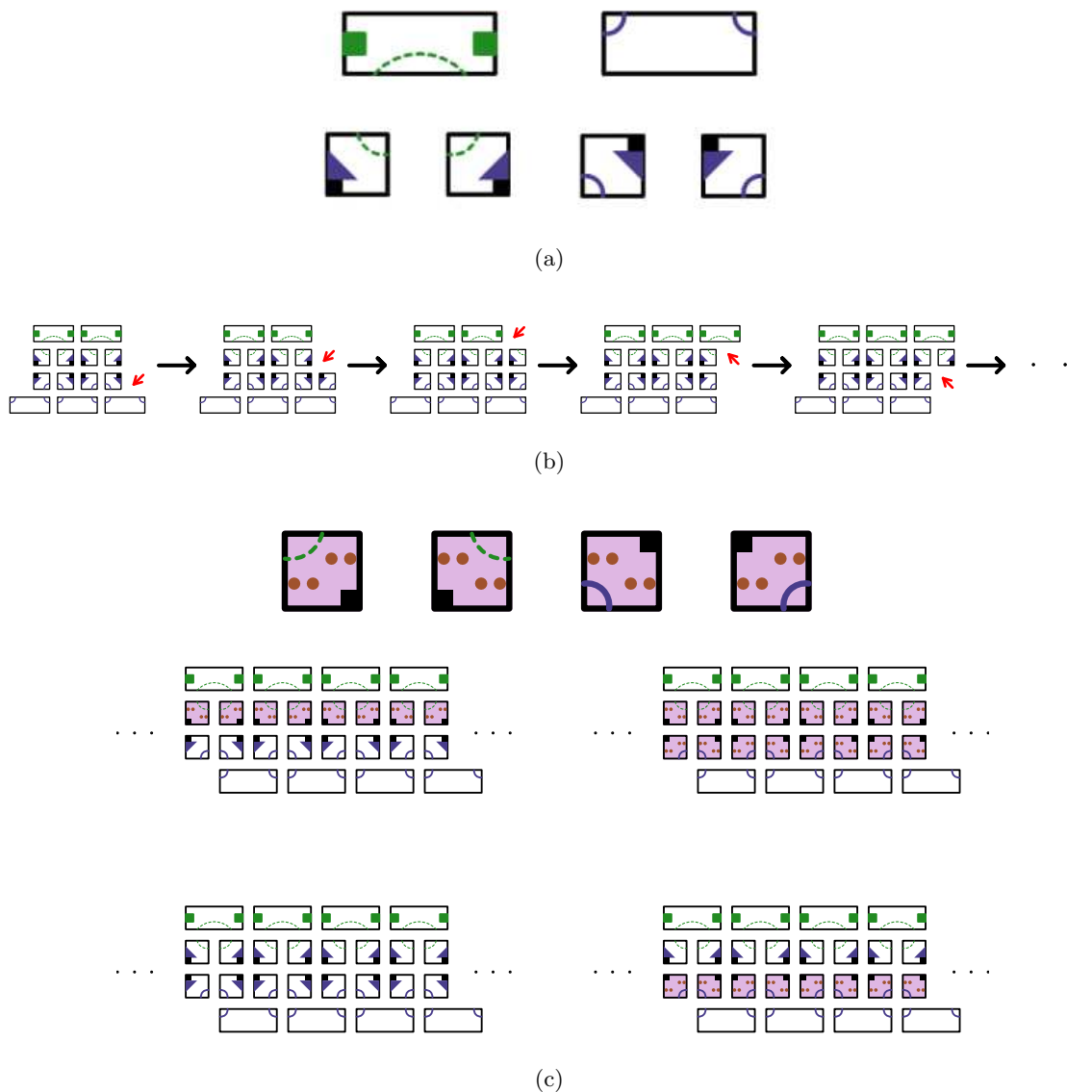


Fig. 2. The zig-zag tile set. (a) The basic width-4 zig-zag tile set consists of six tile types. Tiles cannot be rotated. The tiles shown here have unique bonds that determine where they fit in the assembly: each label has exactly one match on another tile type. (b) The zig-zag tiles are designed to form the assembly shown here. Tiles can attach to the crystal where they match two edges of the crystal. Two alternating tiles in each column enforce the placement of the double tiles on the top and bottom, ensuring that growth continues in a zig-zag pattern. While growth on the right end of the molecule is shown here, growth occurs simultaneously on both ends of the molecule. At each step, a new tile may be added at the location designated by the small arrow. (c) The tile set shown in Figure 2(a) forms only one kind of assembly. The addition of the four tiles shown here allows four types of assemblies to be formed. Each assembly grows by copying its sequence (the vertical cross-section of tiles).

to layer. As an example, the tile set shown in Figure 2(c) has two tiles for each position and therefore can propagate one of four strings. This construction can be generalized to an additional number of bits by adding

tiles to the tile set in Figure 2(c) – 2^n sequences can be propagated by a tile set containing $4n + 2$ tiles. Further, a related tile set containing only 6 tiles can copy binary sequences of arbitrary width.

The growth of a crystal increases the number of copies of the original information present in the ribbon but does not produce any new growth fronts, so copying occurs at a constant rate. This information copying can be accelerated by periodic forces that cause ribbons to break. For each new ribbon that is created by breakage, two new growth fronts become available. Repeated fragmentation will therefore exponentially amplify an initial piece of information. Occasionally, a tile matching only one bond rather than two will join the assembly, resulting in a copying error, which will also be inherited. Such copying errors, inevitable in any physical implementation of tiles (e.g. [32]), will lead to evolution if ribbons with certain sequences grow faster than others.

3 Dynamics of Crystal Evolution

In this section we formulate a simple dynamical model to determine whether crystal growth and breakage leads to selective amplification, and to elucidate which properties of the environment and tile set are important in determining the replication rate of a sequence. The model tracks the concentration of crystals of each possible sequence. For a sequence s , two parameters are of interest: F_s , the number of growth fronts that can copy sequence s , and R_s , the number of columns of tiles, totaled over all crystals, with sequence s . The number of columns, R_s , increases at a rate proportional to the number of growth fronts, F_s , times the rate at which a new column can be added to a growth front, k_s . As this model is meant to be used in cases where growth rates depend upon the sequences s , k_s depends on s , reflecting the influence of tile set and environment. New growth fronts are produced when assemblies split into two pieces. We'll assume that splitting occurs with equal probability at each column, at a splitting rate p_s per column. (Again, this might be sequence-dependent.) Crystals die at rate f by being flushed out of solution.

Assuming that the growth rate is greater than the splitting rate ($k_s > p_s$), the dynamics of this system can be described by two linear differential equations for each sequence.

$$\frac{d}{dt} \begin{bmatrix} R_s \\ F_s \end{bmatrix} = \begin{bmatrix} -f & k_s \\ p_s & -f \end{bmatrix} \begin{bmatrix} R_s \\ F_s \end{bmatrix} \quad (1)$$

Because mutations are not included in this model, pairs of equations describing the growth and replication of a sequence are decoupled from equations describing the dynamics of other sequences. It is therefore not difficult to solve them; for a given sequence, the eigenvalues of the solution are $-f \pm \sqrt{k_s p_s}$. Assuming the death rates for all sequences are the same, sequences with faster growth rates and higher splitting rates are therefore amplified more than sequences that grow and split more slowly. As the death rate increases, only the sequences for which $\sqrt{k_s p_s}$ is large remain in solution. These sequences are selected for.

How does crystal evolution compare to RNA or DNA replication, e.g. of viral or bacterial genomes [12], in which the growth rate is proportional to the concentration of sequences? As the decaying eigenmode dies away, $R_s \rightarrow \sqrt{\frac{k_s}{p_s}} F_s$. (The ratio $\sqrt{\frac{k_s}{p_s}}$ can be interpreted as half the average length of growing and splitting crystals. The ratio is halved because each crystal has R_s rows and two growth fronts.) A new equation that measures only the dynamics of F_s , assuming $R_s \approx \sqrt{\frac{k_s}{p_s}} F_s$, is

$$\frac{d}{dt} F_s = \sqrt{k_s p_s} F_s - f F_s \quad (2)$$

With the addition of mutation, where a mutation from sequence s to sequence t happens at rate m_{st} , this model is equivalent to Manfred Eigen's model of RNA replication [12] with the replication rate of a sequence replaced by the parameter $\sqrt{k_s p_s}$:

$$\frac{d}{dt} F_s = \left(\sqrt{k_s p_s} - f \right) F_s + \sum_t (m_{ts} F_t - m_{st} F_s) \quad (3)$$

Thus, this model is a generalized version of Eigen’s model, with DNA or RNA replication being the special case where $k_s = p_s$. When $k_s > p_s$, the fitness ($\sqrt{k_s p_s}$) of a crystal in a given environment depends on both the growth rate and the splitting rate. Thus, to show that a particular sequence s is fit, we must therefore show that $\sqrt{k_s p_s}$ is large, and that for unfit sequences t , $\sqrt{k_t p_t}$ is small.

4 A Zig-Zag Ribbon Metabolism

By what basis might some zig-zag crystals grow faster than others? In some models of crystal growth [32], the rate of attachment of a tile to a crystal is proportional to the concentration of the tile in solution, and the rate at which a tile is removed is related to the energy loss due to breaking the bonds between the tile and the rest of the crystal. Thus, the growth rate of crystals can be made faster by increasing the concentration of their component tiles in solution. Increasing growth rates increases a crystal’s fitness. Similarly, increasing breakage rates also increases a crystal’s fitness (unless breakage occurs more often than growth).

In previous examples of zig-zag ribbon evolution [25], tile sets (or “chemistry”) needed to be made more complicated in order to achieve more complex selection. In contrast, in biology a single chemistry has led to the evolution of more and more complex organisms. Such evolution has occurred because the fitness of a biological organism is a function of whether its genome contains a program that efficiently directs resource acquisition, development or relations to other organisms in its environment, and thus changes in the environment can drive the evolution of complexity (and visa versa).

Is there an analog of this mechanism for crystal evolution? That is, is there a single tile set that allows zig-zag crystals to achieve selective advantage in many different environments by performing functions adapted to their environment? While a zig-zag ribbon cannot direct any function except its own assembly, the fact that the assembly of tile crystals can perform universal computation [31] suggests that the answer could be “yes”.

An important element of the survival of a self-replicating system in the physical world is the ability to handle variation in the availability of raw materials needed for growth. While modern cells use genetic networks and signal transduction in order to respond optimally to available raw materials, here we describe how zig-zag crystals could evolve a simple “metabolism” by using tile assembly to compute which tiles to use for growth. This mechanism, consisting of a set of tile types and their environment, is too complex to be a model of real crystal growth processes. It is instead intended as a conceptual demonstration that it is possible for zig-zag crystals to evolve complex phenotypes.

We consider a situation where tile resources may be limited and where the addition of a tile may provide information about the environment. In the examples presented here, boundary tiles are present at varying concentrations while the concentrations of non-boundary tiles do not vary.

Two types of boundary tiles, called measurement tiles (Figure 3(a)), initiate new rows from the right during upward growth. Both measurement tiles have the same input edge, but different measurement tiles have different output edges. Because measurement tiles share the same input edges, both available measurement tiles can attach to the right edge of the crystal. The chance that a particular measurement tile will attach is dependent on its concentration in solution. As will be described below, which measurement tile attaches determines the output edge that guides proceeding assembly of the crystal. We will assume that while the relative concentrations of measurement tiles change, their total concentration stays the same.

Another set of tiles may also become more or less available as time goes on. These are the resource tiles (Figure 3(a)). Changes in the concentrations of resource tiles may be correlated with changes in the concentrations of measurement tiles. While both measurement tiles have the same input edge, each resource tile has a different input edge. Only the resource tile that matches the left label on the binding site where a resource tile may be added can fit.

Figure 3(a) shows a set of “computation tiles” that perform boolean functions. In Section 2, we described how the requirement that a tile match the perimeter of an assembly by two edges in order to attach allows tile assembly to copy a sequence. A similar principle allows a crystal to perform local computations that modify the sequence: the two edges by which the tile attaches to the assembly serve as inputs, and the two edges of the tile that remain unattached serve as outputs [31]. These outputs then serve as inputs for future

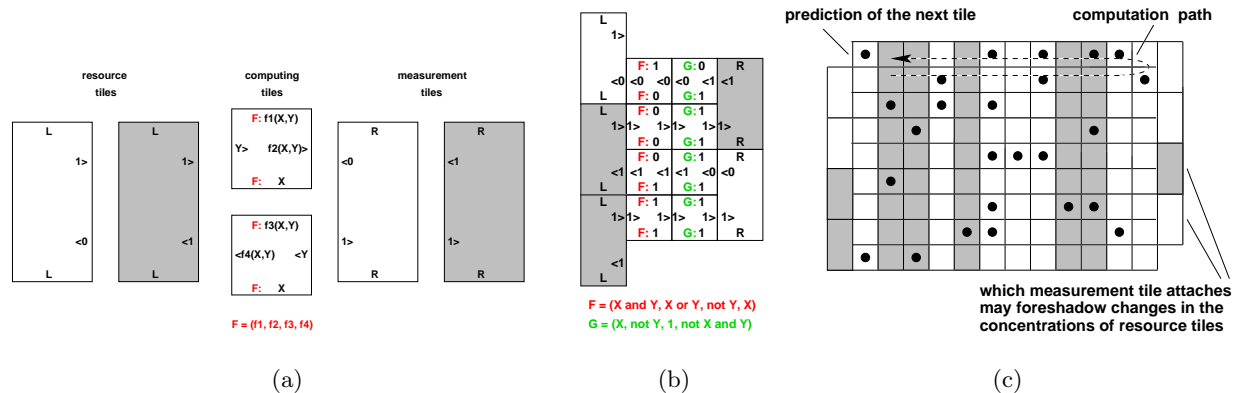


Fig. 3. Zig-zag crystals which exhibit metabolic control. (a) A tile set for the evolution of metabolic control contains computation, resource and measurement tiles. Computation tiles (middle) contain two kinds of labels: those on the left and right encode a boolean value, either 0 or 1, (shown as X, Y, or $f(X,Y)$) and the direction of assembly, either left or right. The labels on the top and bottom encode a gate type (**F** is shown) and a boolean input and output. Each gate type consists of four boolean functions : those passed to the top on leftward and rightward growth, and those passed to the left and right on leftward and rightward growth respectively. A computation tile has the same gate type on the top and bottom, thus ensuring that the sequence of gate types is copied from row to row. We assume that gate types corresponding to all combinations of four boolean functions are provided: there are 8×16^4 such tiles, which is admittedly quite large, although similar behavior should be possible with many fewer tile types. Resource tiles are boundary tiles to the left that have the same output value, a 1, but different input values. A crystal must provide binding sites for common resource tiles in order to grow quickly. Measurement tiles have the same input, but different outputs. These outputs can provide growing tile assemblies with information about the environment. “Smart” crystals use the information provided by the measurement tiles to predict which kind of resource tile is most available. (b) An example computation using the gates shown in (a). For each gate type, the four computations in parenthesis are the four computations for each gate type, in the order shown on the computation tiles in (a). (c) A large assembly consisting of the tiles in (a). The shading of computation tiles represents their gate type, which is passed from row to row and determines the genotype of the ribbon. Squares with dots represent tiles that output 0’s, and squares with no dots represent tiles that output 1’s. The 0’s and 1’s represent the state of a computation, which is updated (but not necessarily copied) from row to row.

computation steps. On the computation tiles shown here, the label on the bottom of each tile encodes the gate type and an input value, and the label on the top encodes the same gate type as well as an output value. The left and right edges of the tile encode input and output values, depending on the direction of assembly. Thus, with each zig and each zag of growth, the sequence of gate types is copied verbatim from row to row, while simultaneously the sequence of boolean values are being processed according to the logic specified by the gates (Figure 3(b)). That is, the computations performed by tile attachment modify the sequence of boolean values from layer to layer, but the sequence of gate types is inherited intact. The concentration of the computation tiles remains constant over time.

The label on the output edge of an attaching measurement tile, either 0 or 1, will serve as an input for the attachment of the next tile, a computation tile. Conversely, the label that will be an input edge for a resource tile, either 0 or 1, will be an output of this series of computations. Thus, the order in which measurement tiles arrive at the right side serves as input to the computation tiles, which in turn produce outputs consisting of a series of input edges for resource tiles on the left side.

What kind of assembly is selected for in this environment? Fit assemblies are those which have the highest replication rate. If it is assumed that all assemblies split at the same rate, a fit assembly is one that grows quickly. Because the environment changes over time, the fittest assembly is the one that has a large *average* growth rate. The rate at which a new row is added, k_s , is the inverse of the total time that is needed to complete a row. The time needed to add each measurement and computation tile stays constant, because

the rate of tile addition is dependent on tile concentration in solution, and these tiles are present in constant supply. (While the concentration of the two kinds of measurement tiles vary, their total concentration remains constant and both types will bind at a given binding site.) The time needed to bind a resource tile changes, however, and is dependent on the current concentration of the resource tile. Two observations can be made. First, a row is added more quickly if fewer computation tiles are used, and second, a row is added more quickly when the resource tile that is needed to complete the row is abundant. To achieve the first requirement, an assembly should be as thin as possible. To achieve the second, an assembly should correctly predict the abundant resource tile and assembly should produce a binding site for it rather than for a less abundant resource tile.

How can a program predict the future concentrations of tiles? The assembly of tiles transforms an input signal, a series of bits received from the output edges of the measurement tiles, to an output signal, a series of binding sites for resource tiles. This transformation depends on the sequence of gate types that is propagated in the crystal. A fit assembly produces an output signal that is the same as the binding site of the more abundant resource tile¹. For example, when the input and output signal are time varying but identical, as in Figure 4(a), an output signal that is the same as the input signal accomplishes this goal. When they are exactly opposite, as in Figure 4(b), inverting the input signal, as is shown, accomplishes this goal. Thus, the most fit genomes in these environments are the empty sequence and the inverting gate respectively.

In some environments, an assembly that successfully predicts the identity of the more abundant resource tile must perform a less trivial computation. Figures 4(c) and 4(d) show examples, described below, of assemblies that do such computations. While a thinner assembly may add the computation tiles in its row more quickly, it would often ask for less abundant resource tiles. Thus, it would spend time waiting to bind these tiles, and therefore might grow more slowly overall than an assembly with more computation tiles that used abundant resource tiles.

A tile program that is well adapted to the environment in Figure 4(c), where the output signal is a time-delayed copy of the input signal, consists of a series of “delay” gates. A delay gate passes the bit it receives from the previous row to the left and passes the bit received from the right to the next row. With a program consisting of n delay units, the crystal will respectively bind a resource tile with a 1 or 0 binding site $2n$ rows after receiving a corresponding measurement tile with a 1 or 0 output site. For a longer or shorter delay, more or fewer delay gates may be used.

When measurement tiles provide no information about the concentrations of resource tiles, as in Figure 4(d), a successful assembly is one that ignores the information received from the measurement tiles and computes a set of outputs in a temporal pattern very similar to the changes in the abundant resource tile. The assembly shown uses its rightmost gate to discard the input from the measurement tiles. It uses a binary counter program [11] (the center two computation tiles) to count to four over and over again. A counter consists of a series of exclusive or / and gates. The inputs to the counter come from rightmost digit of the counter (which is a 1 at each iteration) and the bottom edge of the counter tiles. At each iteration, the counter has two outputs – an output to the left which signals to the output tile to the left, and a set of outputs that become inputs to the next iteration of the counter. The “counter” receives its name because these outputs are, read as a binary number, one larger than the inputs. When the counter reaches its maximum value, in this case three, a one is output to the left. The left-most gate uses this periodic trigger to change its requested resource tile. The assembly shown asks for four 1-type resource tiles, then asks for four 0-type resource tiles, and repeats this cycle. If the requests are in tandem with the periodic change of resource tile type, the assembly will spend little time waiting for resource tiles, and will grow quickly.

The assemblies containing these programs will spend little time waiting for resource tiles; thus, they will grow faster than other assemblies of the same size that must wait for the right resource tile to become avail-

¹ In growth that proceeds downward, rather than upward, the tiles labelled resource tiles function as measurement tiles, and vice versa. Thus, both assemblies that can predict the resource tile types that are available from the measurement tiles, and those that can predict the measurement tile types that will be available from the current concentration of resource tiles will be selected for. Note, however, that gate types may be non-deterministic during downward growth, which could result in crystal growth stalling when a tile is incorporated that creates a binding site that matches no gate tile’s outputs. Therefore, consideration of downward growth rates is necessary for a complete evaluation of a crystal’s fitness; but we neglect it here to simplify the presentation.

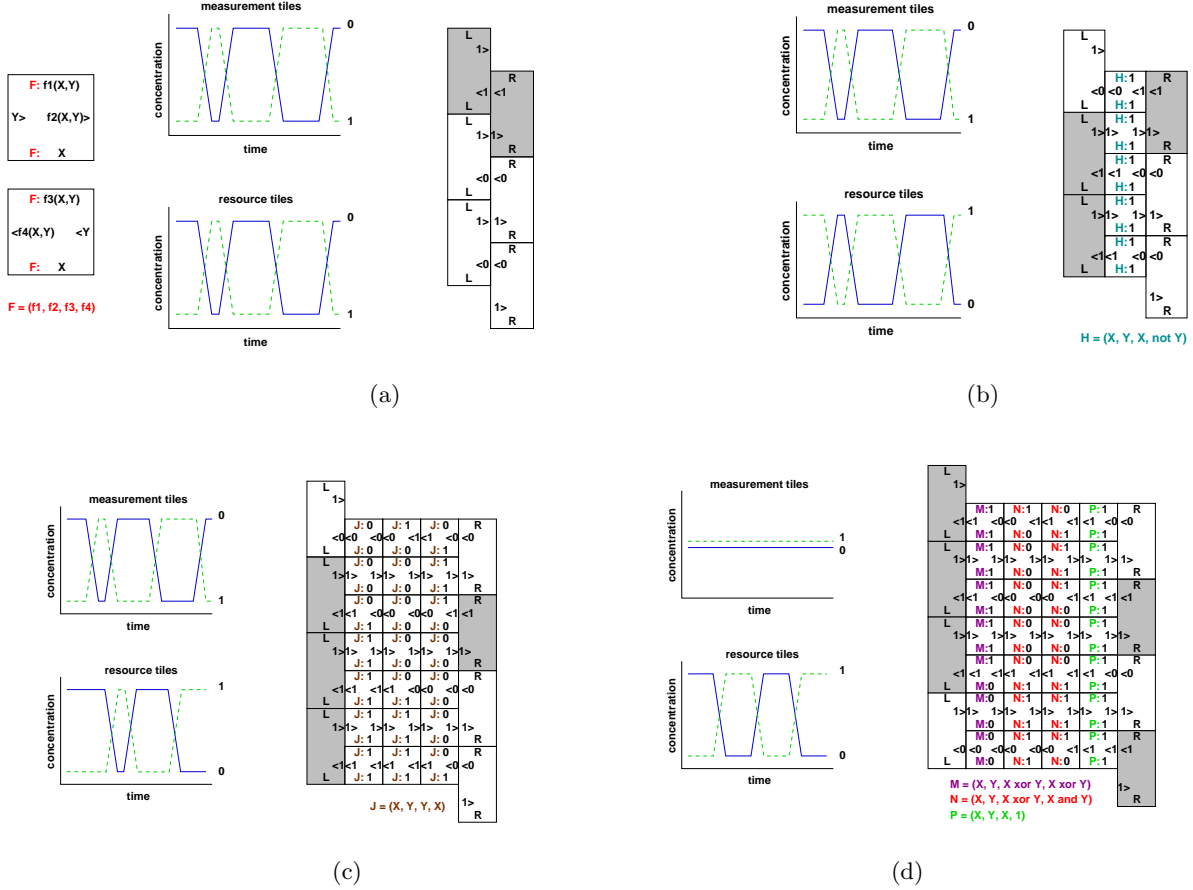


Fig. 4. Time varying tile concentrations and adapted assemblies. Assemblies that grow from a set of tiles that encode boolean functions. Shown here are four environments in which the concentrations of tiles change over time, and an assembly that is well adapted to each environment. Selection favors crystals that correctly predict the concentration of sometimes scarce resource tiles. (a) Over time, measurement tiles and resources tiles have exactly the same time-varying concentrations. A well-adapted assembly asks for a resource tile with the same label as the measurement tile that was received. (b) Opposite resource and measurement tiles have the same concentrations. A well-adapted assembly uses a single computation tile that inverts the input from the measurement tile, so that the opposite resource tile can bind. (c) The kind of measurement tiles that are abundant is perfectly correlated with the type of resource tiles that will be abundant after a fixed time delay. A well-adapted assembly stores information about previous tiles and passes it across the assembly. If the time necessary to pass the type of measurement tile across the assembly is approximately the same as the delay in the correlation, the assembly will ask for the abundant resource tiles. A wider assembly produces a longer delay. (d) Measurement tiles provide no information, but resource tiles change regularly in a way that an assembly can predict. The assembly shown here uses a small counter [11] to change its resource tile request every 8 layers. The left-most row remembers the current choice until the counter sends the message to change it. The rightmost column disregards the information provided by the measurement tiles.

able. But will they grow faster than thinner assemblies, which don't have to spend time adding computation tiles, even if they sometimes wait for resource tiles? In the examples provided, the answer is yes, if the total concentration of resource tiles is sufficiently low.

As an illustration, we compare the growth rate of the matched delay assemblies of Figure 4(c) with the growth rate of the “null assembly”, shown in Figure 4(a), both growing in the delay environment of

Figure 4(c). Under what circumstances would a delay assembly of width k grow faster than the null assembly? If tiles are added at an average of 1 per second, and resource tiles overall are D times less common than computation and measurement tiles, the growth rate of a perfectly synchronized delay assembly of width k “J” gates is $2k + D + 1$ per zig-zag. (A width k assembly is suited to a delay of $\Delta t = k(2k + D + 1)$ seconds.) Assuming that the type of resource tile that is abundant switches on average every s seconds, with $\Delta t \gg s$, the null assembly adds a zig-zag every $2(D + 1)$ seconds on average². When $2k < D + 1$ (and Δt is such that the assembly provides the right delay), the delay assembly grows faster. Clearly, this is true for large enough D .

Similarly, for the binary counter assembly (Figure 4(d)), the growth rate of the perfectly synchronized counter of width k is $2k + D + 1$ seconds per zig-zag³. The growth rate of the null assembly is $D + 1$ seconds per zig-zag half the time for an average growth rate of $2D + 2$ seconds per zig-zag. Thus, when $k + D + 1 < 2D + 2$, the counter assembly is more fit. These examples illustrate that when D is large, that is, when the time spent adding computation and measurement tiles is negligible compared with the time spent waiting for resource tiles, performing the right computations can make an assembly more fit.

In order for a crystal that is good at predicting the current availability of resource tiles to be fit, it must be able to pass along its fitness to its descendent crystals. In this section, we have shown how every row of a crystal could contain the information for a simple program. Thus, when a crystal splits, each descendent contains the same program, which will also be executed. However, while the program is preserved by the descendents, the state of the program is not. The descendent crystals may start by executing the program from many steps ago. In some such cases, such as where a crystal is running a program to keep track of delays, the descendent crystal would then not grow well until it resynchronizes with the environment.

It might seem easier for an assembly to simply accept both kinds of prediction tiles. However, the chemistry of the tile set forbids this – 1 and 0 labels do not match, and therefore cannot bind to each other. The fitness landscape is such that increases in fitness can only be achieved by good predictions. Thus, our examples address the question we set out to consider: how a fixed chemistry (a tile set) induces a selection pressure in which crystals must compute in order to be fit.

5 Evolution of Universal Sensing and Response

In the last section, we showed how a set of tiles that allow crystals to execute and replicate a program could, in the right environment, select for crystals performing a useful computation. However, the tiles shown in Figure 3(a) cannot simulate a Turing machine, and therefore cannot perform universal computation [26]. Thus there are computations the tiles cannot express; such computations might be needed for accurate prediction in some environments. In this section, following [24], we describe a tile set that can simulate a Turing machine and how we can alter this tile set to perform the sensing of measurement tiles and binding of resource tiles described in Section 4. Because the Turing machine can compute any desired function, crystals grown using this tile set are capable of universal sensing and response. That is, in response to arbitrarily complex relationships between measurement and resource tiles, the fittest programs can also become arbitrarily complex.

A Turing machine consists of a long work tape which has a sequence of symbols written on it, and a head that can be in a finite number of states. The head examines the work tape, one symbol at a time, executing a series of movements and updates to the symbols written on the work tape based on what it observes locally. At each state of the computation, the head is in a particular state and at a particular position on the work tape. The state and the symbol written on the work tape determine a symbol the head will replace the

² This estimate assumes that since $\Delta t \gg s$, at any particular time the resource and measurement tile concentrations are uncorrelated. So half the time, the resource and measurement tiles are compatible, and the crystal grows at a rate of $D + 1$ seconds per zig-zag; and half the time the resource and measurement tiles are not compatible, and the crystal essentially doesn’t grow.

³ Note that counters whose natural period is slightly less than the period of the environment will easily remain synchronized with the environment, even when there is variation in the arrival times of the tiles being added.

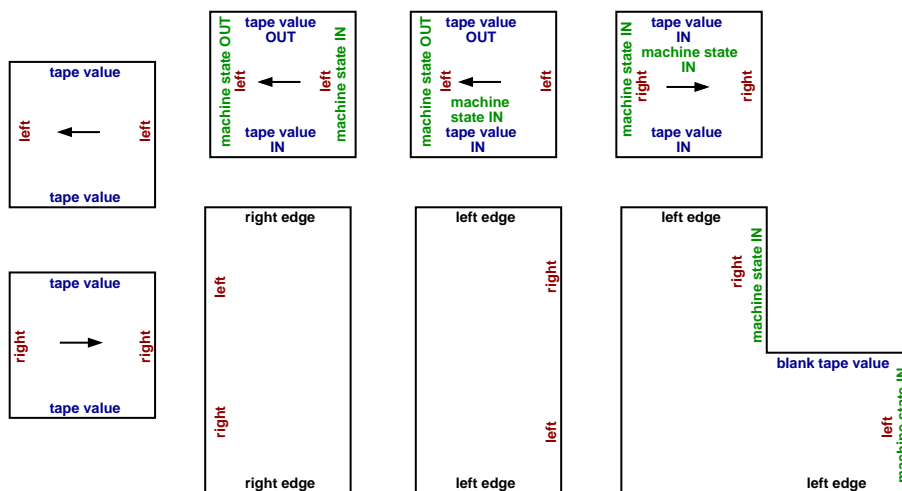


Fig. 5. Tiles for the simulation of a Turing machine. A schematic for a set of tiles whose assembly can simulate the computation of a particular Turing machine on a tape consisting of a row of tiles. A tile type exists to copy each possible work tape value for the cases when assembly is proceeding both to the left and to the right. For each head state and input combination, tile types are needed for encoding the cases where the head is continuing to move in the same direction as the last step, is switching directions on the next step, or has switched directions on the last step. For head state and work tape combinations where the head should move the right after computation, the directions of assembly will be the reverse of those on the tiles shown here. Left and right double tiles form the boundary of the work tape. When more work tape is needed to the left, an L-shaped tile can provide one extra work tape location on the next row (converse tiles exist to extend the work tape to the right). For a Turing machine that has k work tape symbols and s head states, $3ks + 2k + 2s + 2$ tiles are needed to simulate it. (A slightly more complex tile set can also shrink the width of the zig-zag when less work tape is needed.)

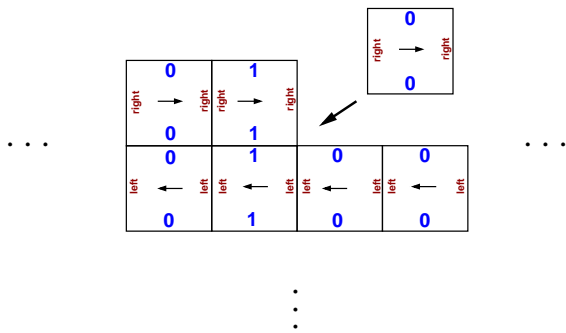
current work tape symbol with, the direction the head will move in (either one step to the left or right) and the next state for the head to enter.

The set of tiles that allows zig-zag ribbon assembly to simulate the execution of a particular Turing machine⁴ are shown in Figure 5. Each row of an assembly of these tiles represents the work tape at a particular time point in the computation. The design of the tile set is such that assembly proceeds up the ribbon, most of the tape is copied from row to row (Figure 6(a)), but some cells are altered as directed by the Turing machine. For compactness and efficiency, all tape manipulations performed during a single unidirectional run of the Turing machine head occur in a single layer of the crystal. Thus, there is one layer per reversal of the Turing machine head.

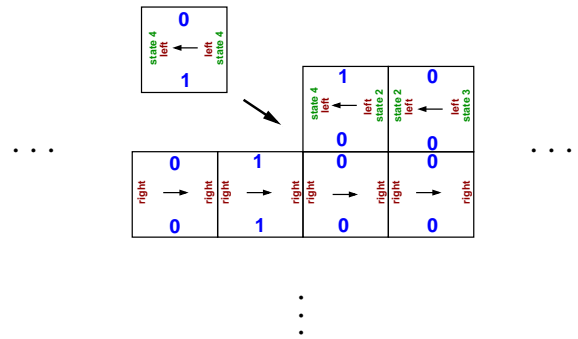
At each step of assembly, exactly one location on the top row of the assembly has a place for a tile to join by two edges, and thus is available for assembly. At each such location, one of three things happens. If the position where the tile is to be added is not the position of the head, the attachment of a new tile will copy the tape from the last row. When assembly is proceeding to the right, the left edge will present a “right” label, and a tile can attach to a location along the top row if it directs assembly to the right and matches the tape value presented by the cell directly beneath it. Likewise, a tile can attach as assembly is proceeding to the left if it matches the “left” label presented by the right edge and matches the tape value presented by the tape (Figure 6(a)). These steps copy the values on the work tape.

At locations where the head is present, values on the work tape can change in subsequent layers. Here, a matching tile type not only matches the old work tape value and the direction of assembly, but also

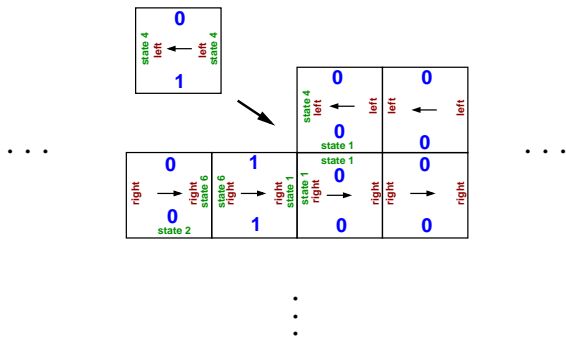
⁴ While zig-zag assemblies copy layers on both growth fronts, for the purposes of our argument, we will assume that computation on zig-zag assemblies proceeds in only one direction (upward). While this is not necessarily so for the tile sets we describe, growth can be restricted to one direction by using a transformed tile set [33] that uses extra tiles to prevent growth in the wrong direction.



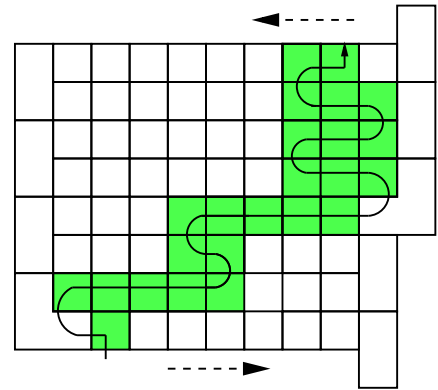
(a) At a position where the head is not present, an attaching tile copies the value on the work tape.



(b) When the head is an input from the right, computation proceeds because the attaching tile matches in the input state of the head. The output edge of this tile contains the next head state. If computation continues to the left, this head state is outputted to the left. The value on the work tape is updated according to the new value specified by the old work tape value and head state.



(c) If a head state and work tape value direct the head to move in the opposite direction as assembly is proceeding, the head state is output up, and computation occurs on the assembly of the next level, by attaching a tile that contains the head state on its bottom edge. Computation can then continue in the opposite direction (or switch directions again.)



(d) An example of the progress of the head (shaded) of a Turing machine on a zig-zag that is simulating a Turing machine. White cells simply copy work tape values from the row beneath them.

Fig. 6. Computation with the Turing machine tile set.

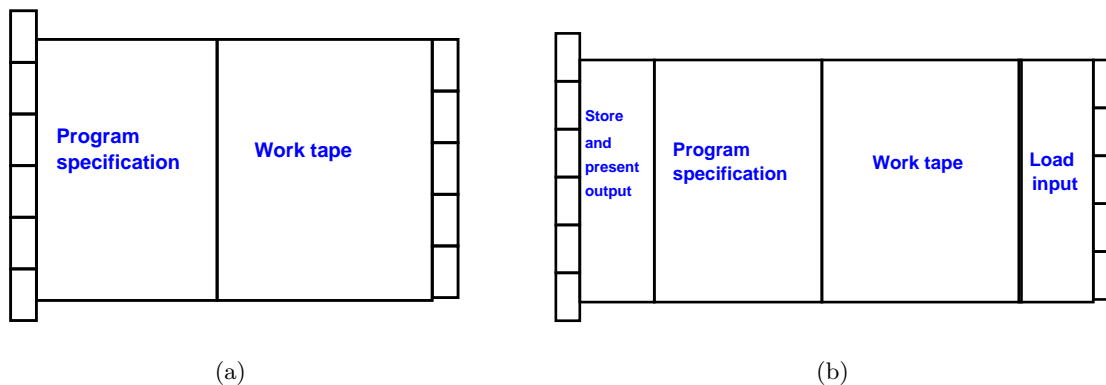


Fig. 8. Using a tile set capable of universal computation for metabolic control. (a) The construction shown in Figure 5 shows how to translate an arbitrary Turing machine into a tile set that simulates it. Applying this construction to a universal Turing machine (UTM) yields a tile set that forms zig-zag ribbons of the type shown here: the program being executed by the UTM is stored in the ribbon separate from the work tape used by the program. (b) A tile set that simulates a universal Turing machine can be combined with boundary tiles that are either resource or measurement tiles. Tiles can also be added to load the input from measurement tiles onto the work tape and to output the desired binding site for the resource tile. With such a tile set, crystals that simulate a Turing machine that correctly predicts the concentrations of resource tiles over time would be most fit.

the current head state. The output edges of such a matching tile direct the new head state and the new value of the work tape at this location (Figure 6(b)). If the tape value and head state direct the head to move in the direction of assembly, the output edge in the direction of assembly (either to the left or right) encodes the new head state. If the tape value and head state direct the head to move in the direction opposite to the direction of assembly, the upward-facing edge of the attaching tile encodes the new head state (Figure 6(c)). Computation continues when assembly finishes in the current direction and zig-zags back to the column where the tile attached (Figure 6(d)).

Because a zig-zag tile set exists for the simulation of an arbitrary Turing machine, such a tile set exists for the simulation of a universal Turing machine. A universal Turing machine is a Turing machine that can simulate the execution of any other Turing machine when both the desired Turing machine and this Turing machine’s initial work tape are encoded on the universal Turing machine’s work tape [26]. Small universal Turing machines exist that compute efficiently [30] and simulate a Turing machine by encoding it on one part of the tape and a work tape on another, as illustrated in Figure 8(a).

It is not hard to imagine using the measurement and resource tiles described in Section 4 instead of the single kind of boundary tiles shown in Figure 5 with the computation tiles that can simulate a universal Turing machine. It would also be possible to add a few tiles to input the measurement value and pass it onto the work tape, and likewise, to pass an output from the Turing machine to the left edge of the assembly as the binding type to present in order to bind a resource tile. The result would be assemblies with the structure shown in Figure 8(b). With such a tile set, it could be possible for assemblies that computed and responded to any desired correlation between measurement and resource tiles to grow. Given an environment in which there is a complex correlation between the measurement and resource tiles that can be computed by a Turing

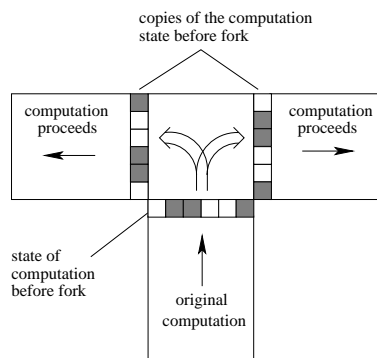


Fig. 7. Reproduction by budding. While splitting is simple, assemblies might also program their reproduction time by “budding” instead of breaking. Budding assemblies may reproduce by first assembling a structure that allows computation to proceed in two directions.

machine, under conditions where the resource tiles are sufficiently rare, the fittest assembly would be one that exactly computed this correlation and asked for the appropriate resource tiles.

If the correlation between measurement and resource tiles were very complex, it is possible that by the time the assembly that exactly simulated the correlation finished computing it, the concentrations of the resource tiles would already have changed. In such an environment, a complex program for prediction may not be selected for. However, one can imagine an almost identical environment where that program would be selected for, in which the time the measurement tiles were present, the wait between presenting the relevant resource tiles, and the time the resource tiles were present were all longer by a constant factor. In an environment where the pattern of increasing and decreasing concentrations of measurement and resource tiles was sufficiently slowed down, the complex assembly would eventually be able to predict the right resource tiles in time, and would therefore be fit.

Since Turing-computable correlations have arbitrarily long history dependence, it may be difficult or impossible for a descendent crystal whose growth edge is at a previous state in the computation to resynchronize after crystal reproduction by fragmentation. A simple augmentation to the tile set rectifies this problem: a special head state triggers a “budding” process that duplicates and forks the computation (Figure 7). This produces two growth fronts that have the exact same computation state as well as program.

A generalization of this type of algorithmically controlled morphogenesis was previously described in detail for investigating the Kolmogorov complexity of self-assembled shapes [28]. Since some shapes that could be created using this tile set may have selective advantage (particularly shear-prone shapes, for example, may be good at reproducing), evolution using such a tile set would result in selection for assemblies that contain programs for certain shapes. Descendants of these crystals will compute the same shape (perhaps starting from a random step in the computation) so that descendent crystals will also have selective advantage⁵.

6 Discussion

While it is possible to increase the complexity of selective pressures by increasing the complexity of tile sets used for growth, we’ve suggested here that tile sets exist that allow crystals to encode and run programs that can predict arbitrarily complex changes in the environment. Evolution using such a tile set as a medium could produce very complex sequences that grow well because they are particularly good at predicting changes in growth conditions. This suggests that crystal growth chemistries logically can support open-ended evolution with arbitrarily complex fitness landscapes.

Our argument that some assemblies will be more fit relies only on their growth rates, which are determined by how effective the assemblies are at predicting the availability of resource tiles. However, as Section 3 illustrates, to be fit, assemblies must not only grow quickly, but also shear frequently. Not enough is known about shearing frequencies to make confident predictions, as these rates are surely dependent on the kind of forces that produce shearing in practice. However, it seems safe to assume that wider zig-zag assemblies would shear less frequently than thin ones in most cases. If this were the case, of two assemblies that predicted the availability of resource tiles equally well, the thinner assembly would be more fit. Such an effect could actually have the effect of favoring correct *and concise* programs for prediction of resource tile concentrations, i.e. it is a natural implementation of Occam’s razor. This scenario is somewhat akin to Solomonoff inference [27] and Levin search [18] in that short programs that produce correct results are found by biased random search.

However, our arguments in this paper are limited: they neglect several important real life features of crystal growth. For example, while the model used in this paper prohibits any tile that matches fewer than two bonds from attaching, such attachments occur at a rate dependent on the physical conditions of assembly. If this error rate is too large, two things can happen. First, particularly large programs that take a long time to run may have difficulty accurately computing without mistakes, and there may be a preference for so-called robust programs (if such programs exist for the tile set used), which compute an answer correctly even with a couple of assembly errors. Second, it may be possible for assemblies that ask for the absent resource

⁵ In cases where a complete shape is necessary for selective advantage, it is also possible to use a construction where crystals can grow forward as well as backward, so that the parent crystal can grow back the piece of the shape that was lost during splitting [33].

tile to eventually attach the available but non-matching resource tile. Also, we have also not considered the effects of backward growth on the fitness landscape of crystals. For some tile sets, backward growth is not deterministic and quickly leads to a configuration that cannot be continued except by making an error, in which case backward growth stalls and can be neglected. For other tile sets, backward growth is deterministic and produces the same class of patterns that can be produced by forward growth. Although many tile sets are intermediate between these extremes, and backward growth should be considered, we expect that there are many tile sets for which crystal evolution can lead to complex fitness landscapes. Yet another simplification we have made is to ignore stochastic effects during assembly and how they affect the time at which an assembly asks for a resource tile. Again, we do not expect such considerations do significantly change our general conclusions.

Despite these and other limitations to our study as presented here, we believe that the qualitative features of the mechanisms that we describe should be preserved in a more realistic model of crystal growth. While the tile sets we described here are too complex to implement experimentally at this time, it is not unreasonable to believe that much simpler tile sets could produce complex adaptations in response to resource limitations. Our initial investigations into whether the ideas here could be tested in experiments suggest that tile sets containing as few as ten tile types could in fact exhibit complex evolution in an environment with constant tile concentrations, as will be described elsewhere. Such a small tile set and environment would be amenable to laboratory experiments to test whether non-trivial crystal genotypes that efficiently use available resources could evolve. While selection for larger and larger genotypes require more and more accurate copying of information [12] and selection based on smaller fitness differences, error-resistant tile sets [34, 8, 21], which contain more tiles but copy more accurately could be used. In principle these tile sets can reduce error rates as much as is desired.

One might think that for the fittest species in a given environment to be arbitrarily complex, the environment must be equally complex. But it is not immediately clear whether this is actually the case. Since there is no fastest algorithm for some functions [3], this would seem to imply that in a crystal growth environment that requires crystals to compute such a function, evolutionary pressure favoring speed would lead to ever-increasing complexity. On other hand, faster programs are in general larger. So while it is possible that evolutionary pressure might favor faster algorithms, the crystal must also be wider in order to contain the complex program and copying the program slows down the crystal's growth. This trade-off is also present in biological, *in vitro*, and artificial evolution.

It is also interesting to ask about the *minimal* requirements for open-ended evolution of crystals. Is it possible that there is a "universal" environment where tile concentrations are constant or vary in a simple periodic pattern, but open-ended evolution is still possible? To answer such a question, it may be worth considering effects that are beyond the simple model of crystal growth and replication kinetics used in this paper. We considered only fitness of species for growth in the exponential phase, where resources are not depleted by other crystals, and where there is no interaction between crystals. Relaxing these assumptions may allow other interesting routes to complexity. For example, could competition between crystals for tiles lead to an "arms-race" of more and more complex strategies for growth? Conversely, might symbiosis between crystal programs lead to interesting phenotypes not explored here? Might a crystal growth chemistry support parasitic crystals that evolve to grow by using existing crystals rather than single tiles? Supporting this last possibility, initial experiments with DNA crystals indicate that joining of crystals does occur [13].

Acknowledgments: We thank Andrew Turberfield, Paul Rothmund, Robert Barish, Ho-Lin Chen, and Ashish Goel for insightful conversations and suggestions. This work was supported by NASA Grant No. NNG06GA50G.

References

1. R. D. Barish, P. W. K. Rothmund, and E. Winfree. Two computational primitives for algorithmic self-assembly: Copying and counting. *Nano Letters*, 5:2586–2592, 2005.
2. M. A. Bedau, J. S. McCaskill, N. H. Packard, S. Rasmussen, C. Adami, D. G. Green, T. Ikegami, K. Kaneko, and T. S. Ray. Open problems in artificial life. In *Artificial Life VII*: 6, pages 363–376, 2000.

3. M. Blum. A machine-independent theory of the complexity of recursive functions. Journal of the ACM, 14:322–336, 1967.
4. N. Bowden, A. Terfort, J. Carbeck, and G. Whitesides. Self-assembly of mesoscale objects into ordered two-dimensional arrays. Science, 276:233–235, 1997.
5. A. G. Cairns-Smith. The origin of life and the nature of the primitive gene. Journal of Theoretical Biology, 10:53–88, 1966.
6. A. G. Cairns-Smith. Genetic Takeover and the Mineral Origins of Life. Cambridge University Press, 1982.
7. A. G. Cairns-Smith and H. Hartman. Clay Minerals and the Origin of Life. Cambridge University Press, 1986.
8. H.-L. Chen and A. Goel. Error free self-assembly using error prone tiles. In DNA Computing 10, Berlin Heidelberg, 2004. Springer-Verlag.
9. A. Chworos, I. Severcan, A. Y. Koyfman, P. Weinkam, E. Oroudjev, H. G. Hansma, and L. Jaeger. Building programmable jigsaw puzzles with RNA. Science, 306:2068–2072, 2004.
10. S. R. Collins, A. Douglass, R. D. Vale, and J. S. Weissman. Mechanism of prion propagation: Amyloid growth occurs by monomer addition. PLoS Biology, 2:e321, 2004.
11. M. Cook, P. W. K. Rothmund, and E. Winfree. Self-assembled circuit patterns. In J. Chen and J. Reif, editors, DNA Computing 9, volume LNCS 2943, pages 91–107, Berlin Heidelberg, 2004. Springer-Verlag.
12. M. Eigen. Self-organization of matter and evolution of biological macromolecules. Naturwissenschaften, 58(10):465–523, 1971.
13. A. Ekani-Nkodo, A. Kumar, and D. K. Fygenson. Joining and scission in the self-assembly of nanotubes from DNA tiles. Physical Review Letters, 93:268301, 2004.
14. D. K. Fygenson, H. Flyvbjerg, K. Sneppen, A. Libchaber, and S. Leibler. Spontaneous nucleation of microtubules. Physical Review E, 51:5058–5063, 1995.
15. M. M. Hanczyc, S. M. Fujikawa, and J. W. Szostak. Experimental models of primitive cellular compartments: Encapsulation, growth, and division. Science, 302:618–622, 2003.
16. G. F. Joyce. Directed evolution of nucleic acid enzymes. Annual Review of Biochemistry, 73:791–836, 2004.
17. M. G. Lagoudakis and T. H. LaBean. 2-D DNA self-assembly for satisfiability. In E. Winfree and D. K. Gifford, editors, DNA Based Computers V, volume 54 of DIMACS, pages 141–154, Providence, RI, 2000. American Mathematical Society.
18. L. A. Levin. Universal sequential search problems. Problems of Information Transmission, 9:265–266, 1973.
19. 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.
20. S. Pitsch, A. Eschenmoser, B. Gedulin, S. Hui, and G. Arrhenius. Mineral induced formation of sugar phosphates. Orig Life Evol Biosph., 25(4):297–334, 1995.
21. J. H. Reif, S. Sahu, and P. Yin. Compact error-resilient computational DNA tiling assemblies. In DNA Computing 10, Berlin Heidelberg, 2004. Springer-Verlag.
22. P. W. K. Rothmund. Using lateral capillary forces to compute by self-assembly. Proceedings of the National Academy of Sciences, 97:984–989, 2000.
23. P. W. K. Rothmund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology, 2:424–436, 2004.
24. P. W. K. Rothmund and E. Winfree. The program-size complexity of self-assembled squares. In Symposium on Theory of Computing (STOC), pages 459–468. ACM, 2000.
25. R. Schulman and E. Winfree. Self-replication and evolution of DNA crystals. In Advances in Artificial Life, 8th European Conference, volume 3630, pages 734–743, Berlin Heidelberg, 2005. Springer-Verlag.
26. M. Sipser. Introduction to the Theory of Computation. PWS Publishing Company, 1997.
27. R. J. Solomonoff. A formal theory of inductive inference: Part 1 and 2. Information Control, 7:1–22, 224–254, 1964.
28. D. Soloveichik and E. Winfree. Complexity of self-assembled scale-invariant shapes. In DNA Computing 10, Berlin Heidelberg, 2004. Springer-Verlag.
29. H. Wang. An unsolvable problem on dominoes. Technical Report BL-30 (II-15), Harvard Computation Laboratory, 1962.
30. S. Watanabe. 5-symbol 8-state and 5-symbol 6-state universal turing machines. Journal of the Association for Computing Machinery, 8:476–483, 1961.
31. 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, RI, 1996. American Mathematical Society.
32. E. Winfree. Simulations of computing by self-assembly. Technical Report CS-TR:1998.22, Caltech, 1998.

33. E. Winfree. Self-healing tile sets. In J. Chen, N. Jonoska, and G. Rozenberg, editors, Nanotechnology: Science and Computation. Springer, 2006.
34. E. Winfree and R. Bekbolatov. Proofreading tile sets: Error-correction for algorithmic self-assembly. In J. Chen and J. Reif, editors, DNA Computing 9, volume LNCS 2943, pages 126–144, Berlin Heidelberg, 2004. Springer-Verlag.
35. 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.
36. M. C. Wright and G. F. Joyce. Continuous in vitro evolution of catalytic function. Science, 276:614–617, 1997.
37. Y. Yokobayashi, R. Weiss, and F. H. Arnold. Directed evolution of a genetic circuit. Proc. Nat. Acad. Sci. USA, 99:16587–16591, 2002.