Nat Comput
DOI 10.1007/s11047-011-9293-6

Programming and evolving physical self-assembling systems in three dimensions

Navneet Bhalla · Peter J. Bentley · Peter D. Vize · Christian Jacob

Abstract Being able to engineer a set of components and their corresponding environmental conditions such that target entities emerge as the result of self-assembly remains an elusive goal. In particular, understanding how to exploit physical properties to create self-assembling systems in three dimensions (in terms of component movement) with symmetric and asymmetric features is extremely challenging. Furthermore, primarily top-down design methodologies have been used to create physical self-assembling systems. As the sophistication of these systems increases, it will be more challenging to use top-down design due to self-assembly being an algorithmically NP-complete problem. In this work, we first present a nature-inspired approach to demonstrate how physically encoded information can be used to program and direct the self-assembly process in three dimensions. Second, we extend our nature-inspired approach by incorporating evolutionary computing, to couple bottom-up construction (self-assembly) with bottom-up design (evolution). To demonstrate our design approach, we present eight proof-of-concept experiments where virtual component sets either defined (programmed) or generated (evolved) during the design process have their specifications translated and fabricated using rapid prototyping. The resulting mechanical components are placed in a jar of fluid on an orbital shaker, their environment. The energy and physical properties of the environment, along with the physical properties of the components (including complementary shapes and magnetic-bit patterns, created using permanent magnets to attract and repel components) are used to engineer the self-assembly process to create emergent target structures with three-dimensional symmetric and asymmetric features. The successful results demonstrate how physically encoded information can be used with programming and evolving physical self-assembling systems in three dimensions.

Keywords Embodied computation · Evolutionary computing · Physical information encoding · Rapid prototyping · Self-assembly · Tile assembly model

N. Bhalla (✉) · P. D. Vize · C. Jacob
Department of Computer Science, University of Calgary, 2500 University Drive N.W., Calgary, AB T2N 1N4, Canada
e-mail: nbhalla@ucalgary.ca

P. D. Vize
e-mail: pvize@ucalgary.ca

C. Jacob
e-mail: cjacob@ucalgary.ca

P. J. Bentley
Department of Computer Science, University College London, Malet Place, London WC1E 6BT, UK
e-mail: p.bentley@cs.ucl.ac.uk

P. D. Vize
Department of Biological Sciences, University of Calgary, 2500 University Drive N.W., Calgary, AB T2N 1N4, Canada

P. D. Vize · C. Jacob
Department of Biochemistry and Molecular Biology, University of Calgary, 2500 University Drive N.W., Calgary, AB T2N 4Z6, Canada

1 Introduction

Self-assembly is identified as a fundamental aspect to achieving embodied computation (Stepney et al. 2005, 2006). Understanding the interplay between programmability/controllability and self-organisation is required in order for engineering emergence in this form of natural

Published online: 01 November 2011

computing (Li et al. 2008). Theoretical systems executing algorithms using self-assembly in three spatial dimensions have been proposed, but the physical construction of components being able to direct the self-assembly process in three dimensions has not been achieved (Pelletier and Weimerskrich 2002). Here we present two methods to create sets of proof-of-concept components with physically encoded information which is used to direct the self-assembly process to achieve three-dimensional target structures.

As seen throughout nature, the plethora of complex inorganic and organic system are the result of self-assembly. Complex self-assembled entities emerge from decentralised components governed by simple rules. Natural self-assembly is dictated by the morphology of the components and the environmental conditions they are subjected to, as well as their component and environment physical and chemical properties—their information (Ball 1999; Thompson 1942, reprinted 1992). Components, their environment, and the interactions among them form a system, which can be described by a set of simple rules. Coupled with this bottom-up self-assembly construction process, bottom-up design is used with living organisms where the process of evolution is displayed through their genetic rule sets—their DNA. Through transcription to RNA and translation to proteins, these rules are mapped to physical shapes representing a transfer of information as described by the central dogma of molecular biology (Crick 1970). Proteins, the resulting three-dimensional shapes, are the primary building blocks of living organisms. One could view entities in nature constituting programs that are based on the rules present in a given system (Wolfram 2002). These programs are the interaction and transformation of physically and chemically encoded information in a given environment. The process of self-assembly has been shown to being equivalent to performing a physical computation (Winfree 1995, 1999), as information is compared and exploited to build larger structures.

However, designing and constructing artificial self-assembling systems, particularly in three spatial dimensions, remains an elusive goal. There are examples of self-assembling systems in three dimensions. However, these examples are limited in being able to create structures with only symmetric features (Douglas et al. 2009; Terfort et al. 1997), using components which are not able to freely move in three-dimensional space (Zykov et al. 2005), relying on templates to direct the self-assembly process (Boncheva et al. 2005; Gracias et al. 2002; Wu et al. 2002), or the creation of open structures (Boncheva et al. 2003a; Gracias et al. 2000). Open self-assembly (Whitesides and Gryzbowski 2002) refers to the creation of structures which do not have a defined boundary, in contrast to closed structures (Whitesides and Gryzbowski 2002). Consequently, understanding how to exploit physical properties to create self-assembling systems continues to be extremely challenging. To compound this challenge, primarily top-down design methodologies have been used to create physical self-assembling systems (Groß and Dorigo 2008). As the sophistication of these systems increases it will be more challenging to use top-down design, due to self-assembly being an algorithmically NP-complete problem (Adleman et al. 2002). How to design a set of physical components and their environment, such that the components self-assemble into a target structure remains an open problem.

In pursuit of addressing this open problem, we continue to build upon our three-level approach for designing self-assembling systems (Bhalla et al. 2007, 2010, 2011; Bhalla and Bentley, in press). The three-level approach comprises of: (1) specifying a set of self-assembly rules, (2) modelling these self-assembly rules to determine the outcome of a system in software, and (3) translating to a physical system by mapping the set of self-assembly rules using physical components and environment properties. Our approach is consistent with the definition of self-assembly (Whitesides and Gryzbowski 2002), refined here as a process which involves components that can be controlled through their proper design and their environment, and which are adjustable (components can adjust their position relative to one another).

Here we extend our three-level approach for designing self-assembling systems in three spatial dimensions. This extension includes the development of new self-assembly rules at level one and new software at level two, which is distinct from (Bhalla et al. 2007, 2010; Bhalla and Bentley, in press) where only self-assembling systems in two spatial dimensions were considered. New components and a new environment that are suitable for three-dimensional self-assembly are required at level three, and follow designs used in (Bhalla et al. 2011). We present how features in these new components, acting as physically encoded information with respect to their corresponding environment, can be exploited to direct the self-assembly process. In particular, we demonstrate how physically encoded information can be used to reduce errors and to create rotational properties during component-to-component interactions. In contrast to (Bhalla et al. 2011), here we use the three-level approach exclusively as a programming approach, and we demonstrate how to direct the self-assembly process to create a variety of target structures by using the different forms of component physically encoded information. In addition, we present a further extension of our three-level approach that incorporates evolving self-assembling systems, by incorporating evolutionary computing which is well suited for addressing NP-complete

problems (Mitchell 1996). Previously we evolved physical self-assembling systems in two dimensions (Bhalla et al. 2010), which we extend here to three dimensions. The three-level approach is inspired by the central dogma of molecular biology, in being able to map a set of self-assembly rules directly to physical shapes. By incorporating evolutionary computing into the three-level approach, it couples bottom-up construction (self-assembly) with bottom-up design (evolution). The evolutionary approach is beneficial in that no knowledge of a target structure's morphology is required, only its functionality.

The next section provides background material to which our programming and evolutionary approach is based upon. Next, the extension of the three-level approach with the incorporation of evolutionary computing to design and create physical self-assembling systems in three dimensions is detailed. Eight experiments follow which demonstrate the creation of programmed and evolved component sets and their translation, via physically encoded information, to physical systems using rapid prototyping (Callicott 2001). We conclude by summarising how this work provides a proof-of-concept means to programming and evolving physical self-assembling systems in three dimensions.

2 Background

Bottom-up self-assembly design (using evolutionary computing or another technique) does not appear to have been achieved in three-dimensional self-assembly at the time of writing. To achieve our objective of being able to evolve self-assembling systems in three dimensions, we present relevant background material on DNA computing (Păun et al. 1998), three-dimensional physical self-assembly, and self-assembly design to support our physical information encoding approach to bottom-up self-assembly design. This relevant literature will also be used to differentiate our contributions to three-dimensional self-assembly.

2.1 DNA computing

Crystallography continues to be an active area in science. One of the most prolific insights was that of Schrödinger and his proposed aperiodic crystal for genetic information encoding, described in his popular science classic *What is Life?* (Schrödinger 1944, reprinted 2003). Schrödinger's aperiodic crystal predates the discovery of the aperiodic structure of DNA by Watson and Crick (1953). Crick acknowledges Schrödinger for providing inspiration to study the structure of DNA (Murphy and O'Neill 1997). In addition to understanding how the aperiodic structure of DNA serves as a genetic information medium in biological systems, research is being conducted on manipulating DNA for nanotechnology and embodied computation (Seeman 1999, 2007). DNA nanotechnology was invented by Seeman (Pelesko 2007), a crystallographer, who realised that three-dimensional lattices could be used to direct molecules simplifying their crystallographic study. The first artificial three-dimensional nanoscale objects, created using self-assembly, were a cube and a truncated octahedron made out of DNA (Chen and Seeman 1991; Zhang and Seeman 1994). However, these objects were not rigid enough to create three-dimensional lattices (Pelesko 2007; Seeman 2004).

Consequently, Adleman (1994) first demonstrated how computing using self-assembling molecules, DNA, was possible. In Adleman's original work, he devised an algorithm to solve a seven node Hamiltonian path problem (Garey and Johnson 1979). Single strands of DNA were created to encode partial sample solutions to the problem (representing the edges and vertices of the directed graph). These partial solutions were then allowed to interact and self-assemble based on Watson–Crick complementarity. The linear double-stranded self-assembled structures representing partial solutions were then filtered using biochemistry and molecular biology techniques to identify the true self-assembled solution.

With the success of Adleman's experiment, research has focused on scaling DNA computing to solve larger problems. As a shift from Adleman's brute-force approach, Winfree developed the abstract Tile Assembly Model (aTAM) to model the self-assembly of molecular structures, such as DNA tiles (Winfree et al. 1998), on a square lattice (Rothmund and Winfree 2000; Winfree 1998a). These tiles use interwoven strands of DNA to create the square body of a tile (double-stranded) with single strands extending from the edges of the tiles. A tile type is defined by binding domains on the North, West, South, and East edges of a tile. Physical DNA tiles were co-created by Winfree and Seeman. DNA tiles are based on mathematical Wang tiles (Wang 1961, 1965), which have been proven to be Turing Universal (Berger 1966; Culik 1996). The aTAM is used to bridge crystallography and mathematical tiling theory using DNA tiles as a method for algorithmic self-assembly (Winfree 1999), where algorithmic and physical information encoding schemes are used to direct the self-assembly process.

In the aTAM, at least one seed tile must be specified to start the self-assembly process. Tiles cannot be rotated or reflected. There cannot be more than one tile type that can be used at a particular assembly location in the growing structure (although the same binding domain is permitted on more than one tile type). All tiles are present in the same environment, a *one-pot-mixture*. Tiles can only bind together if the interactions between binding domains are of

sufficient strength (provided by a strength function), as determined by the *temperature* parameter. The sum of the binding strengths of the edges of a tile must meet or exceed the temperature parameter. For example, if the temperature parameter is two, at least two strength-one bonds must be achieved to assemble a tile, i.e. the temperature parameter dictates *co-operative bonding*. The seed tile is first placed on the square lattice environment. Tiles are then selected one at a time, and placed on the square lattice if the binding strength constraints are satisfied. The output is a given shape of fixed size, if the model can uniquely construct it.

The aTAM has been used to study algorithmic self-assembly complexity, including the Minimum Tile Set Problem (MTSP) (Adleman et al. 2002). The goal of MTSP is to find the lowest number of tile types that can uniquely self-assemble into a target structure. The MTSP is an NP-complete problem for general target structures. A decision problem *C* is NP-complete if *C* is in the class NP (condition one) and every problem in the class NP is reducible to *C* in polynomial time (condition two) (Sipser 1997). Condition one can be met by proving that *C* can be verified in polynomial time. To meet condition one, Adleman et al. (2002) created an algorithm called Unique-Shape that decides whether a tile system uniquely produces the shape of a target structure, which they proved can be verified in polynomial time. Condition two can be met by proving that an already know NP-complete problem can be reduced to *C* (i.e. proving that *C* is NP-hard). The decision problem for determining if the variables in a Boolean formula can be assigned in such a way to have the formula evaluate to true is referred to as satisfiability (SAT) (Sipser 1997). SAT was the first decision problem known to be in the class NP-complete (Cook 1971). One of several special cases of SAT is where the formulae are in conjunctive normal form (CNF) (Sipser 1997). The 3CNF-SAT problem (determining if each clause in a formula is limited to at least three literals) is known to be in the class NP-complete (Karp 1972; Sipser 1997). To meet condition two, Adleman et al. (2002) reduced the 3CNF-SAT problem to the MTSP by encoding a 3CNF-formula into shapes using two structures: (1) a tree, for which it is possible to compute the minimal tile set in polynomial time, and (2) a tree substructure, for which can only be satisfied if and only if the tree substructure can be assembled using distinct tile types. By proving that the MTSP is in the class NP and that the MTSP is in the class NP-hard (by proving that 3CNF-SAT can be reduced to the MTSP), Adleman et al. (2002) proved that the MTSP is in the class NP-complete for general target structures.

In addition to the MTSP, Adleman et al. (2002) also devised the Tile Concentration Problem (TCP). The goal of TCP is to find the relative concentrations of tile types that self-assemble into the target structure using the fewest assembly steps. The algorithmic complexity of TCP has only been calculated for specific classes of target structures. The aTAM and variations of tile-based self-assembly have also been used to investigate self-assembly complexity classes (Jonoska and McColm 2009) and using self-assembly to solve NP-complete problems (Brun 2008a, b).

The aTAM has also been used to study Turing Universal computation via self-assembly. Winfree proved that the aTAM is Turing Universal in two spatial dimensions at temperature two (Winfree 1995). The original version of the aTAM has been extended to three dimensions, using cube-based tiles (Winfree 1998b). It has been proven that the aTAM is Turing Universal in three spatial dimensions at temperature one (Cook et al. 2011). Furthermore, aTAM has been used to study new algorithmic proposals to performing mathematical operations, in both two and three dimensions. In particular, a theoretical algorithm to perform the cyclic convolution product requires three-dimensional DNA tiles (Pelletier and Weimerskrich 2002). However, the physical creation of three-dimensional DNA tiles does not appear to have been achieved at the time of writing.

2.2 Three-dimensional physical self-assembly

Rothmund (2005, 2006) pioneered a technique to create two-dimensional DNA structures, referred to as DNA Origami. This technique uses smaller, linear strands of DNA to act as scaffolding, which self-assemble to a longer, linear strand of DNA folding it into a target structure. The use of folding in DNA Origami has also been extended to three dimensions (Douglas et al. 2009). Only one example was shown in Douglas et al. (2009) where the components created using three-dimensional DNA Origami would then self-assemble into a three-dimensional target structure. In this example, all the components were identical and self-assembled into target symmetrical tetrahedrons.

Physical self-assembly does not need to be at the nanoscale. Both mesoscale (micrometer to millimetre) and macroscale (centimetre and above) offer a unique quality in comparison to molecular and nanoscale self-assembly, in that components at these scales offer flexibility in design and access to types of functionality unparalleled in molecular and nanoscale systems (Boncheva et al. 2003b; Whitesides and Boncheva 2002, 2005). However, three-dimensional self-assembly is limited in being able to create target structures with only symmetric features (Douglas et al. 2009; Terfort et al. 1997), using components which are not able to freely move in three-dimensional space (Zykov et al. 2005), relying on templates to direct the self-assembly process (Boncheva et al. 2005; Gracias et al. 2002; Wu et al. 2002), or the creation of open structures (Boncheva et al. 2003a; Gracias et al. 2000) based on relevant literature.

Of these three-dimensional mesoscale and macroscale examples, (Terfort et al. 1997) demonstrates in one system where all the components used are not identical (two components, one with a *key* shape and one with a *lock* shape bond together to form a cylinder), and (Gracias et al. 2002) demonstrates in one system where the faces of a self-assembled cube had patterned faces (four metallic squares near each corner of a face). In (Gracias et al. 2002), the resulting cubes were created from a self-assembling folding technique, where the faces of the cube are initially attached in a two-dimensional arrangement. Furthermore, the resulting cubes in (Gracias et al. 2002) are not able to self-assemble into large structures, and consequently, the patterning on the faces of the cubes do not direct the self-assembly process. Physical examples of manipulating patterning on components for error-encoding and directing the self-assembly process have been developed using macroscale, physical components in two spatial dimensions (Bhalla et al. 2007, 2010; Bhalla and Bentley, in press). Theoretical examples have been developed at the nanoscale, where the patterning sequences of nucleic acids are manipulated in DNA for error-encoding and directing the self-assembly process as well (Kari and Mahalingam 2010).

2.3 Self-assembly design

As with the three-dimensional self-assembly examples listed above and further examples based on relevant literature (Groß and Dorigo 2008), primarily top-down design methodologies have been used to create physical self-assembling systems. As the sophistication of self-assembling systems increases, it will be more challenging to use top-down design, due to self-assembly being an algorithmically NP-complete problem. Evolutionary computing is well suited for such problems (Mitchell 1996), and offers the potential for a bottom-up design approach to self-assembly. Evolutionary computing has been applied in theoretical studies, where chemical compounds were evolved (Buchanan et al. 2008), and where different cooperative bonding mechanisms were evolved in the context of the aTAM (Terrazas et al. 2007). Physical examples, where evolutionary computing is used in bottom-up self-assembly design has been demonstrated using robotic units (Ampatzis et al. 2009) and macroscale components (Bhalla et al. 2010) in two spatial dimensions.

3 Three-level approach and evolution

Although there has been progress, the design and creation of self-assembling systems remains challenging. To provide an alternative design approach to self-assembling systems, we developed the three-level approach (Bhalla et al. 2007, 2010, 2011; Bhalla and Bentley, in press). The three phases in our approach are: (1) definition of rule set, (2) virtual execution of rule set, and (3) physical realisation of rule set (Fig. 1). The motivation for the three-level approach is in finding the fundamental information structures and rules that enable self-assembly in theory (level one), testing and refining those self-assembly rules through simulation (level two), and testing and refining those self-assembly rules through embodied physical experiments (level three). The three-level approach provides a bottom-up method for designing physical self-assembling systems. This is achieved by being able to directly map a set of self-assembly rules to a physical system. As a result, evolutionary computing can be incorporated into the three-level approach to evolve level one self-assembly rules. Results from the level two modelling are used for evaluation by the evolutionary algorithm (Fig. 1). After running the evolutionary algorithm, the level one rules can be mapped to a physical system if the desired results are achieved.

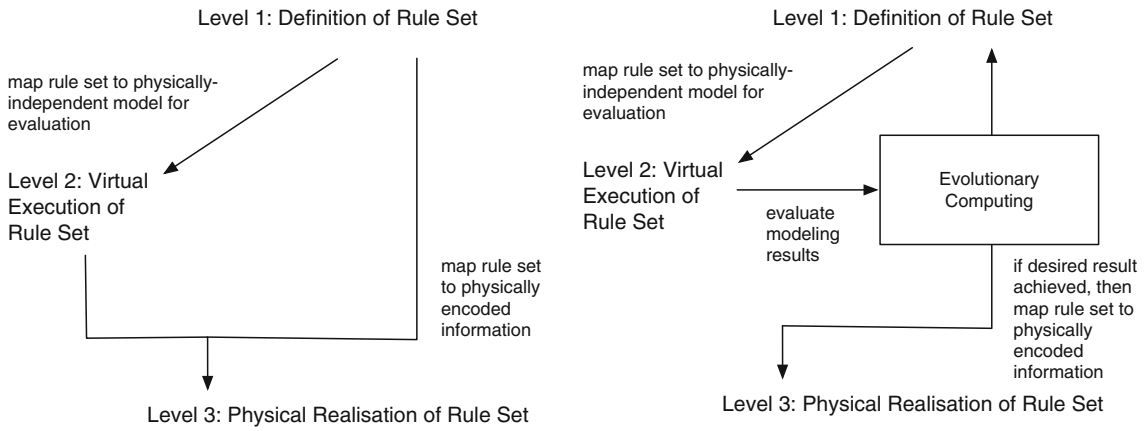


Fig. 1 Three-level approach (*left*) and incorporating evolutionary computing (*right*) (Bhalla et al. 2010)

A *programming* method (which use the three-level approach exclusively) and an *evolutionary* method (which incorporates evolutionary computing) for designing and creating self-assembling system in three dimensions is provided in this section. These two design methods using physically encoded information have the potential to create target structures with symmetric and asymmetric features. Furthermore, these two design methods use components which can move freely in three dimensions, and do not rely on templates in the environment to direct the self-assembly process. The self-assembly design problem we are concerned with is a combination of the MTSP and the TCP in three dimensions (including other constraints, which are described in Sect. 3.4.3).

3.1 Level one: definition of rule set

The specification of self-assembly rules is conducted at level one. The objective of these rules is to provide an abstract, high-level description of self-assembling systems. There are three categories of self-assembly rules, which define a system: *component*, *environment*, and *system* rules.

3.1.1 Component rules

Component rules specify primarily shape and information. Components are similar in concept to DNA tiles. Components are all cubes of unit size. Each face of a component serves as an information location, in a six-point arrangement, i.e. Top–Left–Bottom–Right–Front–Back. Information is abstractly represented by a capital letter (A–L in this example). Furthermore, a subscript (using numbers 1–4) is used with each capital letter (e.g. G_4) to indicate the orientation of that information on a component's face (Fig. 2). If no information is associated with an information location, neutral (meaning where no assembly can take place), the dash symbol (–) is used. The spatial relationship of this information defines a component's type (Fig. 2).

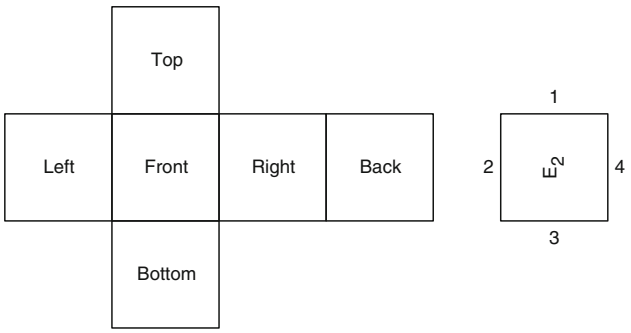


Fig. 2 Spatial relationship defining a component (*left*), and example of information orientation on a component's face (*right*)

3.1.2 Environment rules

Environment rules specify environmental conditions such as the *temperature* of a system and *boundary* constraints. The temperature defines the threshold to which the assembly protocol must satisfy in order for assembly bonds to occur. Components are confined due to the environment boundary, but are permitted to translate and rotate in three spatial dimensions, and interact with one another and their environment. As a consequence and further differentiation to the aTAM, components are permitted to be reflected and to have rotational properties associated with their assembly information locations. An environment temperature of one is used in this implementation.

3.1.3 System rules

System rules specify the quantity of each component type, and component-to-component information interactions (i.e. assembly interactions) and component-to-environment interactions (i.e. transfer of energy and boundary interactions). In this implementation there are two types of system interaction rules referred to as *fits* rules and *breaks* rules. Abstractly, if two pieces of complementary information come into contact (i.e. they fit together, A fits B), it will cause them to assemble. This rule type is commutative, meaning if A fits B, then B fits A. Furthermore, fits rules encapsulate component-to-component rotational interactions. A subscript (using 360, 180, and 90) is used to represent if the faces of complementary components can fit together in either four, two, or in one way respectively (e.g. A fits_{360} B). Abstractly, if two assembled pieces of information experience a temperature above a certain threshold, their assembly breaks. The system rules used in this implementation are specified in Table 1.

3.2 Level two: virtual execution of rule set

At level two, a self-assembly rule set is mapped to an abstract model. We present an extension to the aTAM, the

Table 1 Component information location labels (A–L) and fits and breaks systems interaction rules ('→' transition, '+' assembly, ';' disassembly, and 'Temp_2' temperature 2)

Fits rule	Breaks rule
A fits_{360} B → A+B	Temp_2 breaks A+B → A ; B
C fits_{360} D → C+D	Temp_2 breaks C+D → C ; D
E fits_{180} F → E+F	Temp_2 breaks E+F → E ; F
G fits_{90} H → G+H	Temp_2 breaks G+H → G ; H
I fits_{90} J → I+J	Temp_2 breaks I+J → I ; J
K fits_{90} L → K+L	Temp_2 breaks K+L → K+L

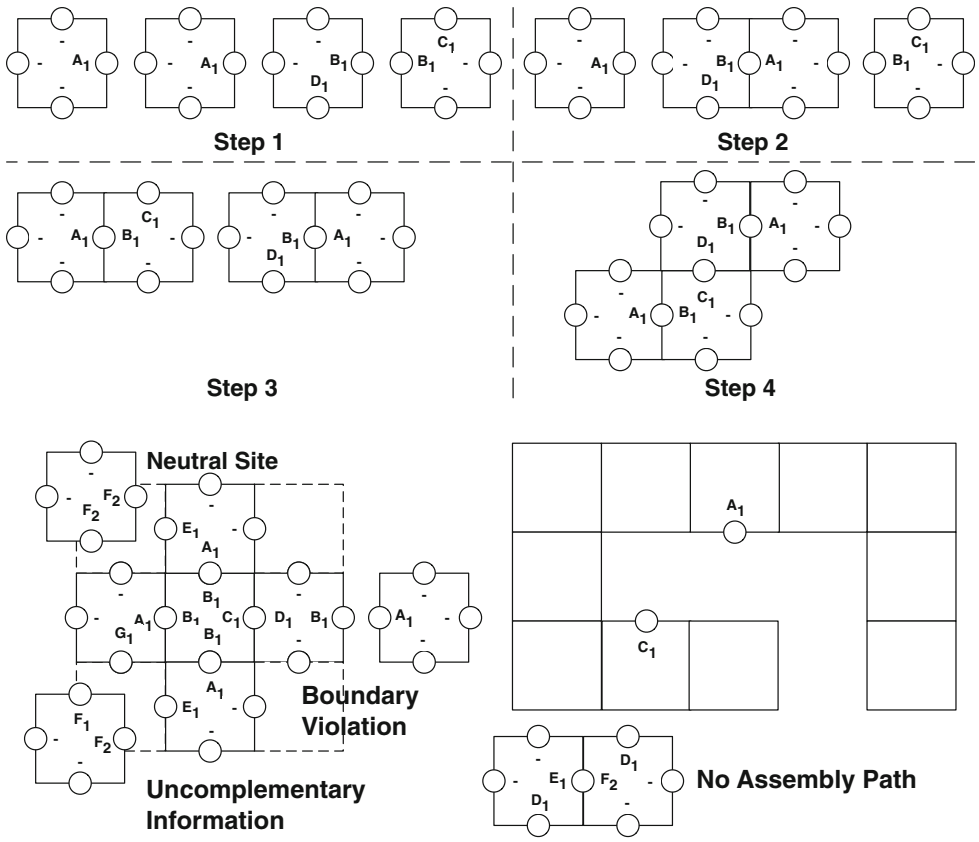


Fig. 3 Example 3DcTAM steps (*top*) and assembly violations (*bottom*; neutral site between F_2 and –, uncomplementary information between F_1 and G_1, boundary violation where the *dashed lines* represent the environment boundary, and where there is no assembly path do to translation and rotation), where a two-dimensional representation is used to represent components in a top view and information labels are in reference to adjacent information locations on a component's face

three-dimensional concurrent Tile Assembly Model (3DcTAM). The 3DcTAM is a modelling technique that is better suited to the type of physical self-assembling systems we focus on (e.g. using components with rotational properties in three spatial dimensions). The 3DcTAM is an extension from two to three spatial dimensions from our original concurrent Tile Assembly Model (cTAM) (Bhalla et al. 2010).

There are five differentiating features to the 3DcTAM from the aTAM and its three-dimensional variant. (1) There are no seed tiles, meaning that any two compatible tiles (components) can start the self-assembly process. (2) Tiles can self-assemble into multiple substructures concurrently. (3) Tiles can be rotated and reflected in three spatial dimensions. (4) More than one tile type can be used at a particular assembly location in a growing structure. (5) All tiles are present in the same environment, one-pot mixture. In this implementation, the temperature parameter is set to one. As a result, 3DcTAM has the capacity of Turing Universal computation at temperature one, similar to how aTAM has been proven to have the capacity of Turing Universal computation at temperature one in three spatial dimensions (Cook et al. 2011).

The initial set of tiles in the 3DcTAM is a multiset (type and concentration). In the 3DcTAM (Fig. 3), a single assembly operation is applied at a time, initialised by selecting a single tile/substructure with an open complementary information location, then the location on the first tile/substructure is labelled *unmatchable*. If there are tiles/substructures with open complementary information locations, all those tiles/substructures are put into an *assembly candidate list*. Tiles and substructures are selected at random (from the assembly candidate list) until a tile/substructure can be added. If no such tile/substructure can be added, due to an *assembly violation* (Fig. 3), then the location is labelled unmatchable. If a tile/substructure can be added, the open assembly locations on the two tiles/substructures are updated, and labelled *match* (all applicable assembly locations, including their rotational properties, must match when adding two substructures). The algorithm repeats, and halts when all assembly locations are set to match or unmatchable. At the conclusion of the algorithm, the resulting structures are placed in a single three-dimensional grid environment to determine if any environment boundary violations occur. A post-evaluation of environment constraints is sufficient for this implementation, as we are more concerned with the set of self-assembled structures than environmental constraints. Alternatively, a post-evaluation of the environmental constraints can be used to determine the necessary dimensions of the environment without hindering the self-assembly of tiles (components).

3.3 Level three: physical realisation of rule set

Level one rules are mapped to a physical system at level three. Here we discuss the component design space (set of physically feasible designs), which is used as part of the mapping process. Component and environment designs follow (Bhalla et al. 2011). However, here we further discuss physical encoding schemes of components in their corresponding environment, by focusing on component physical encoding schemes to prevent errors during component-to-component interactions and to direct the self-assembly process to achieve target structures.

3.3.1 Component design space

The component design space is a combination of an assembly protocol space and a shape space. A 5-magnetic-bit pattern defines the assembly protocol space (Fig. 4). The five permanent magnetic discs can be added to a face of a cube-based component. Of the 32 total 5-magnetic-bit patterns, there are 6 unique complementary pairs of patterns when considering planar rotation of a component's face (Fig. 4). These six codes encapsulate rotational information for component-to-component interactions, where two pairs encapsulate 360°, one pair encapsulates 180°, and three pairs encapsulate 90° rotational component-to-component interactions.

A key–lock–neutral concept defines the shape space (Fig. 5). Either a key, lock, or neutral shape is used on each face of a component. A pair of complementary key and lock shapes is used to create stable joints between complementary components. A 5-magnetic-bit pattern is placed within a key or a lock. However, the magnets are not flush with the surface, creating an air gap, to allow for selective binding between components by maintaining component-to-component interactions to be adjustable (Whitesides and Gryzbowski 2002). In this context, neutral means where no assembly is possible. To simplify the evaluation of correct self-assembly bond formations at the conclusion of the self-assembly process (visually by the user), coloured circles are used (when possible) on neutral sites to identify neighbouring 5-magnetic-bit patterns (referred to as interaction markers). The design of the components were discovered through extensive preliminary trials and experiments conducted by the authors, as well as extensions to two-dimensional component designs (Bhalla et al. 2010). Component specifications are detailed in the Appendix.

3.3.2 Physical component encoding schemes

Two examples are provided to demonstrate how the assignment of the 5-magnetic-bit patterns to keys and locks, as well as a further manipulation to the 5-magnetic-bit

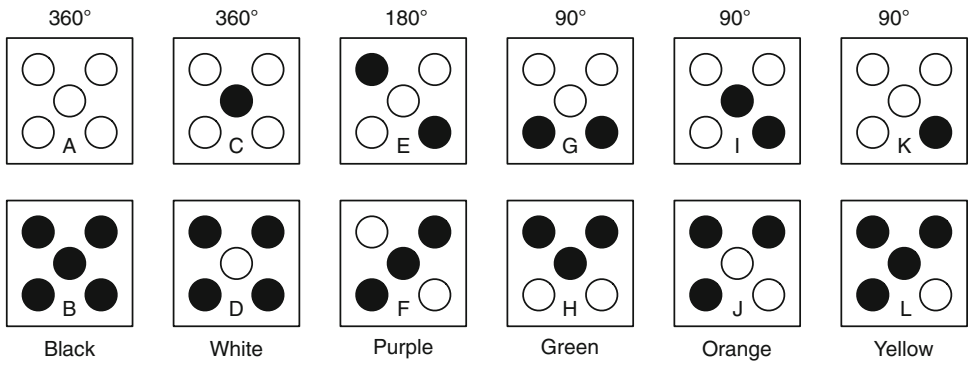


Fig. 4 Six pairs of unique 5-magnetic-bit patterns, where a *closed circle* represents magnetic north and an *open circle* represents magnetic south, along with rotational properties (360°, 180°, and 90°) and *colours* associated with each pair (used as paint markers on components to indicate information associated with component-to-component interactions)

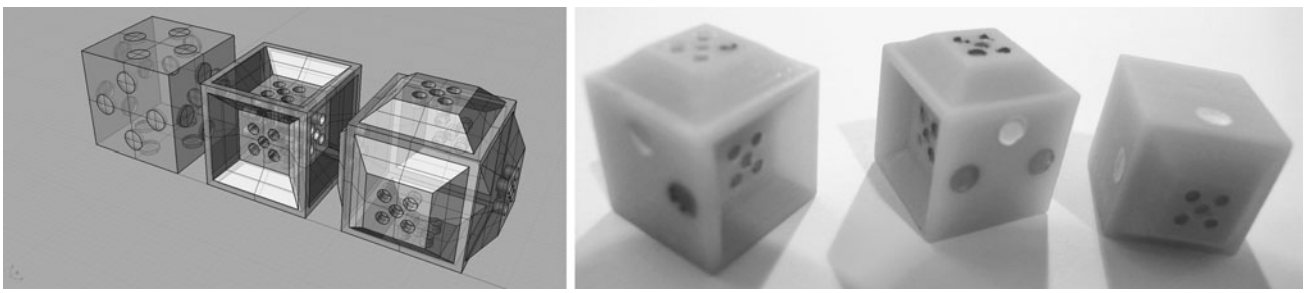


Fig. 5 Component shape space (*left*; showing the base component with information markers, the minimum component with lock shapes on all six faces, and the maximum component with lock shapes on all six faces), and example components (*right*; a component's base shape is 15 mm^3 and a maximum height of 20 mm)

patterns, can be used to reduce component-to-component interaction errors and to direct the self-assembly process. In the following two examples, 1 represents magnetic north and 0 represents magnetic south arbitrarily. A linear representation of the magnets on a component's face are listed as: centre, bottom-right, bottom-left, top-left, and top-right.

3.3.2.1 Minimise component key-to-key error interactions In this first example, a single permanent magnet is used in each location of the 5-magnetic-bit patterns. Using this magnetic condition, the 5-magnetic-bit patterns can be assigned to keys and locks to reduce key-to-key error interactions (Table 2). In this example, the worst possible match between magnets that can occur is a three out of five match (e.g. D interacting with F).

3.3.2.2 Minimise component key-to-lock error interactions In this second example, a single permanent magnet is used in each location of the 5-magnetic-bit patterns which are assigned to keys, and two permanent magnets are used in each location of the 5-magnetic-bit patterns which are assigned to locks. This magnetic condition is used to take advantage of lock-to-lock interactions not being possible, and allows for the reduction of key-to-lock error interactions (Table 3). Although key-to-key interactions will still occur, the environment temperature can be adjusted to break magnetically weak key-to-key interactions and maintain magnetically strong key-to-lock interactions. Furthermore, there is more than one configuration of 5-magnetic-bit patterns assigned to keys and locks to reduce key-to-lock error interactions. The configuration in Table 3 is better suited for use in the programming experiments and evolutionary experiments as component-to-component interactions can be reduced with respect to the features of the target structures (which are provided in Sect. 4).

The corresponding physical environment information to these two component physical encoding schemes consists of a jar to provide a boundary for components, the placement of the jar in a rack on an orbital shaker to provide energy to the system (a parameter in setting environment temperature), and mineral oil in the jar to allow components to freely move in three spatial dimensions. Mineral oil is used to prevent the permanent magnets from corroding, and to provide the appropriate viscosity. The design of the environment was discovered through extensive preliminary trials and experiments conducted by the authors. Environment specifications are detailed in the Appendix.

3.4 Evolving self-assembly rule sets

The objective of the evolutionary algorithm is to search for a *good enough* solution, i.e. a component set (type and concentration) able to self-assemble into a single target structure. Environment and system (fits and breaks) rules are fixed. The following is an overview of the evolutionary algorithm used, genotype and phenotype representations, fitness function, and selection, crossover, and genetic operators used.

3.4.1 Evolutionary algorithm

A generational evolutionary algorithm (Mitchell 1996) is used. The evolutionary unit, *gene*, is a single component. A *databank* of gene sequences (linear representation of the Top–Left–Bottom–Right–Front–Back faces, using A–L with subscripts 1–4 to indicate information orientation on a face, and the '–' symbol) is used to identify and compare genes. There are 1,291,467,969 total and 53,977,737 unique genes (when considering three-dimensional shape and rotations). In the following experiments, 10,000 generations and a population size of 100 individuals was used. Elitism is used, where the top 10% of individuals are copied to the next generation. The parameter settings of

Table 2 Key/lock designations to 5-magnetic-bit patterns with component information location label (A–L), and system interaction rules ('→' transition, '+' assembly, ';' disassembly, and '$Temp_2$' temperature 2), to reduce key-to-key error interactions

Key/lock	5-Magnetic-bit	Label	Fits rule	Breaks rule
Lock	00000	A	A $fits_{360}$ B → A+B	$Temp_2$ breaks A+B → A ; B
Lock	10000	C	C $fits_{360}$ D → C+D	$Temp_2$ breaks C+D → C ; D
Lock	01010	E	E $fits_{180}$ F → E+F	$Temp_2$ breaks E+F → E ; F
Lock	01100	G	G $fits_{90}$ H → G+H	$Temp_2$ breaks G+H → G ; H
Lock	11000	I	I $fits_{90}$ J → I+J	$Temp_2$ breaks I+J → I ; J
Lock	01000	K	K $fits_{90}$ L → K+L	$Temp_2$ breaks K+L → K ; L
Key	11111	B	B $fits_{360}$ A → B+A	$Temp_2$ breaks A+B → A ; B
Key	01111	D	D $fits_{360}$ C → D+C	$Temp_2$ breaks A+B → A ; B
Key	10101	F	F $fits_{180}$ E → F+E	$Temp_2$ breaks F+E → F ; E
Key	10011	H	H $fits_{90}$ G → H+G	$Temp_2$ breaks H+G → H ; G
Key	00111	J	J $fits_{90}$ I → J+I	$Temp_2$ breaks J+I → J ; I
Key	10111	L	L $fits_{90}$ K → L+K	$Temp_2$ breaks L+K → L ; K

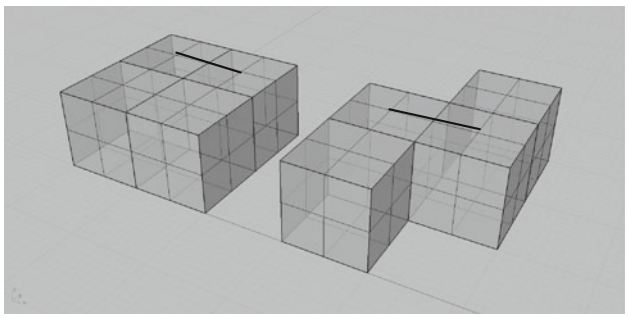
Table 3 Key/lock designations to 5-magnetic-bit patterns with component information location label (A–L), and system interaction rules ('→' transition, '+' assembly, ';' disassembly, and '$Temp_2$' temperature 2), to reduce key-to-lock error interactions

Key/lock	5-Magnetic-bit	Label	Fits rule	Breaks rule
Lock	00000	A	A $fits_{360}$ B → A+B	$Temp_2$ breaks A+B → A ; B
Lock	10000	C	C $fits_{360}$ D → C+D	$Temp_2$ breaks C+D → C ; D
Lock	01010	E	E $fits_{180}$ F → E+F	$Temp_2$ breaks E+F → E ; F
Lock	10011	H	H $fits_{90}$ G → H+G	$Temp_2$ breaks H+G → H ; G
Lock	00111	J	J $fits_{90}$ I → J+I	$Temp_2$ breaks J+I → J ; I
Lock	10111	L	L $fits_{90}$ K → L+K	$Temp_2$ breaks L+K → L ; K
Key	11111	B	B $fits_{360}$ A → B+A	$Temp_2$ breaks A+B → A ; B
Key	01111	D	D $fits_{360}$ C → D+C	$Temp_2$ breaks A+B → A ; B
Key	10101	F	F $fits_{180}$ E → F+E	$Temp_2$ breaks F+E → F ; E
Key	01100	G	G $fits_{90}$ H → G+H	$Temp_2$ breaks G+H → G ; H
Key	11000	I	I $fits_{90}$ J → I+J	$Temp_2$ breaks I+J → I ; J
Key	01000	K	K $fits_{90}$ L → K+L	$Temp_2$ breaks K+L →+K ; L

10,000, 100, and 10% were determined from preliminary experiments conducted by the authors. Parameter settings in the preliminary experiments of 5,000, 50, and 10% respectively, as used in (Bhalla et al. 2010) for evolving two-dimensional self-assembling systems did not lead to good solutions in the three-dimensional case. However, parameter settings of 10,000 generations and 100 individuals per generation did lead to good solutions.

3.4.2 Genotype and phenotype representations

An individual's genotype representation is a variable length list of genes (Fig. 6). At least two genes define a genotype (since this is the minimum for self-assembly to occur). An individual's phenotype representation is the resulting set of self-assembled structures. A single genotype representation may have more than one phenotype representation, depending on the set of components and



(-,-,-,-,C_1,-) (-,-,-,-,C_1,-) (-,D_1,-,-,F_1,-) (-,-,-,D_1,E_1,-)

Fig. 6 Example genotype where *brackets* are represented as (Top, Left, Bottom, Right, Front, Back) (*bottom*), and where the 180 rotational component-to-component interaction for information E and F (represented as a *dark line*) can lead to two possible phenotypes as a result of self-assembly

assembly steps. As a worst-case example, a genotype that consists of n components with information A on all faces and n components with information B on all faces would result in at most $2n!$ phenotypes. Consequently, it is not practical to test all the resulting phenotypes corresponding to a large genotype. Therefore, each individual (genotype) is evaluated three times, at each generation, to help determine the fitness of an individual.

3.4.3 Multi-objective fitness evaluation

A multi-objective fitness function is used to evaluate each individual. The eight objectives can be categorised into evaluating a *general* and a *refined* solution (Fig. 7). The general solution has six objectives: (1) volume (V), (2) surface area (S), (3) mean breadth (B), (4) Euler (E), (5) z-axis, and (6) matches. The volume, surface area, mean breadth, and Euler are calculated using Eqs. 1–4, where n_3 is the number of cubes (components), n_2 is the number of faces, n_1 is the number of edges, and n_0 is the number of vertices. Together, these six general objectives are sufficient to describe the three-dimensional *shape* of the target structures. The volume, surface area, integral mean curvature, and Euler (*connectivity* of a shape) are calculated using 3D Morphological Image Analysis (Blasquez and Poiraudeau 2003; Michielsen and de Raedt 2000; Soille 2003). The second moment of inertia in the z-axis (Jr. et al. 2009) is calculated to identify either identical structures or different structures which have similar reflected features. To distinguish between reflected structures, the number of matching components between a self-assembled structure and the target structure is calculated. A refined solution is accounted for by using two objectives: (7) locations, and (8) error. We consider a refined solution as one that minimises the number of remaining open assembly locations (for the creation of closed target structures), and

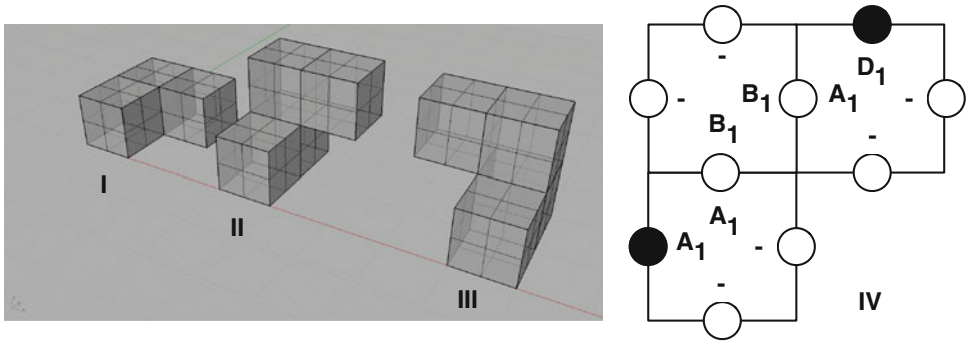


Fig. 7 Fitness objective examples: structure I ($n_3 = 3$, $n_2 = 16$, $n_1 = 28$, and $n_0 = 16$), structures II and III have the same moment of inertia, the number of matches between structures II and III which have similar reflected features is 3, the number of open locations is 2 (indicated using *black circles* in IV; two-dimensional top view corresponding to Fig. 3), and the potential error score in IV is 8 (Table 4)

Table 4 Magnetic error interactions matrix (in reference to Table 3), where the numbers are the sum of all errors between information in the four orientations (using the number of magnets, i.e. two or one, in each mismatched location), and where N/A is in reference to lock-to-lock interactions not being possible

	A	B	C	D	E	F	G	H	I	J	K	L
A	N/A											
B	0	0										
C	12	32	N/A									
D	N/A	8	0	0								
E	N/A	36	N/A	24	N/A							
F	36	24	24	24	6	16						
G	24	24	36	16	24	24	16					
H	N/A	24	N/A	36	N/A	24	21	N/A				
I	36	24	12	32	36	16	24	24	12			
J	N/A	24	N/A	12	N/A	36	24	N/A	27	N/A		
K	12	32	36	24	24	24	16	36	20	30	12	
L	N/A	12	N/A	24	N/A	24	36	N/A	30	N/A	27	N/A

potential assembly errors (due to magnetic interactions). Errors in magnetic interactions is applied to all potential two-component key-to-lock and key-to-key (all five magnets positions are considered in key-to-key errors, and partial interactions are not accounted for) in a system. The potential magnet error is calculated as the sum of the scores in Table 4, where each cell in the matrix is the sum of errors occurring between two pieces of information in all for orientations. The combination of these two objectives also reduce the number of unique components required, as well as favouring 5-magnetic-bit patterns with higher rotational freedom.

$$V = n_3 \tag{1}$$

$$S = -6n_3 + 2n_2 \tag{2}$$

$$B = (3n_3 - 2n_2 + n_1)/2 \tag{3}$$

$$E = -n_3 + n_2 - n_1 + n_0 \tag{4}$$

Each objective is normalised, using the highest and lowest fitness scores from the current generation (Bentley and Wakefield 1997). This method has been shown to be an effective way to calculate multi-objective fitness (Corne and Knowles 2007). For objective i (where i varies from 1 to 6) the average normalised objective (ANO_i) over three 3DcTAM evaluations is calculated and compared to the target objective (TO_i) value. For objective seven, the normalised average over the three 3DcTAM evaluations (ANO_7) is calculated. For objective eight, the normalised objective (NO_8) is calculated with respect to a genotype. The average is not used with objective 8 as the error calculated is based exclusively on a genotype, and does not vary over the three 3DcTAM evaluations. The objectives are then weighted and summed to give a final fitness score F (Eq. 5). The weights were selected from preliminary experiments, and based on experiments conducted by the authors using similar objectives that are appropriate for self-assembling systems in two spatial dimensions (Bhalla et al. 2010). It was found that a weighting where the refined objectives accounted for more than 10% of the overall fitness function did not lead to good solutions.

$$F = \left(0.15 \sum_{i=1}^{6} |TO_i - ANO_i|\right) + 0.05(ANO_7 + NO_8) \quad (5)$$

3.4.4 Selection, crossover, and genetic operators

The fitness scores for each individual are used during selection. Roulette-wheel selection is used to select two parents (favouring lowest fitness scores). The two parents, using a variable-length crossover operator, are used to create two children. Each common gene (determined by the gene databank) between the two parents is copied to each child. Each uncommon gene, for example the gene from parent one, has a 90% probability of being copied to child one (likewise for parent two and child two). After crossover is performed to create two children, the genetic operators duplication, deletion, and mutation are applied to each child. There is a 10% probability of a single gene, chosen at random, of being duplicated, and likewise being deleted. For each information location in a gene, there is a 10% probability of being mutated (equal probability A–L in all four orientations, and –). The parameter settings for crossover, duplication, deletion, and mutation were determined based on experiments in two spatial dimensions that use the same parameter settings conducted by the authors (Bhalla et al. 2010).

4 Experiments and results

We present eight experiments to demonstrate how self-assembling systems can be designed in three dimensions, using the three-level approach as a programming design method (five experiments) and incorporating evolutionary computing into the three-level approach as an evolutionary design method (three experiments). The programming experiments are used to test our three-dimensional physical information encoding scheme. The evolutionary experiments are used to test the three-level approach for the bottom-up generation of self-assembling systems using the functionality of the target structures.

4.1 Programming three-dimensional self-assembling systems

Our hypothesis for the programming experiments was, given the attributes of a target structure, information encoded in the components can be used to enable those components to self-assemble into a three-dimensional target structure. The three-level approach was used to test our hypothesis. A target structure was assigned to each experiment (Fig. 8). For each experiment, enough components were supplied to create up to three target structures. Ten trials were run for each experiment. A virtual trial (level two) was evaluated as successful if all three target structures were created. A physical trial (level three) was evaluated as successful if at least one target structure was created. The difference in the number of created target structures for successful evaluation is due to physical self-assembly being more challenging than modelling. The experimental procedure and results for the programming experiments are described in terms of the three-level approach.

4.1.1 Level one: definition of rule set for programming experiments

These five target structures were chosen since they offer degrees of complexity in terms of the number of components and their concentration, and symmetric/asymmetric features in the target structures. Consequently, the five target structures cannot be created by using the underlying

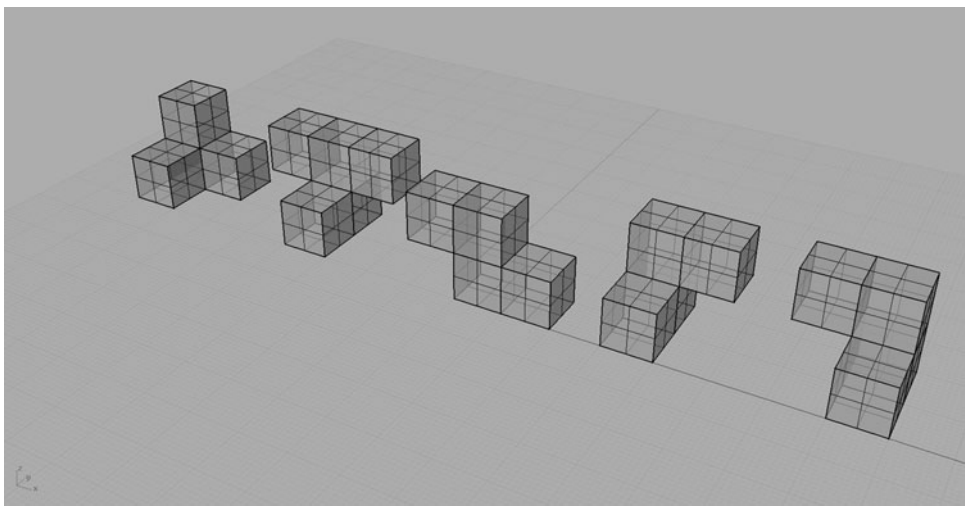


Fig. 8 Target structures for the programming experiments (experiments 1–5 from *left to right*)

cubic-grid pattern of component formations exclusively. Therefore, it is appropriate to determine if the information encoded in the components is sufficient to achieve the target structures by self-assembly.

In these experiments, the independent variable is the set of components, defined by their type and concentration. The dependent variable is the resulting set of self-assembled structures. For each experiment, a designed component set is specified along with a randomly generated component set, in order to test the independent variable. For the designed component sets, component types and concentration were specified to create a single target structure. After this initial set was specified, the quantity of each component type was multiplied by three in order to increase the chances of being able to create up to three examples of the target structures. For the randomly generated component sets, component types were specified by selecting with uniform probability the information (A–L and their orientation 1–4 on a component's face, and –) assigned to each information location. Then the quantity of each component type was multiplied by three in order to increase the chances of being able to create up to three examples of the target structures. This method for creating the designed and randomly generated component sets was chosen as part of our experimental set-up with the aim of providing proof-of-concept evidence to our programming approach, by conducting a statistical significance test (details provided in the level two and three experimental results). A summary of the component rules used for each experiment is provided in Table 5.

4.1.2 Level two: virtual execution of rule set for programming experiments

3DcTAM was used to evaluate the ability of each self-assembly rule set (designed or random) to create its respective target structures. The level two experimental set-up and results are provided.

4.1.2.1 Level two: experimental set-up The component sets from Table 5 were mapped to an abstract representation for 3DcTAM. Each component's shape was a unit cube. The size of the environment was represented as 4 by 4 by 4 units (width, depth, and height). These environment dimensions are related to the physical dimensions of the environment as a ratio between the physical components and environment, and rounded down to the nearest unit. Since 3DcTAM selects tiles/substructures at random to step through the self-assembly process, a different random seed was used to initialise 3DcTAM for each trial. Ten trials were conducted for each experiment.

4.1.2.2 Level two: experimental results Each designed component set successfully created three of their applicable target structures (Fig. 8). These results show that even without a tile (component) acting as a seed, it is still possible to create multiples of the same target structure, when appropriate component information is used. In contrast, none of the randomly generated component sets successfully created at least one target structure, in each experiment. The reasons for the unsuccessful randomly generated component sets include components having uncomplementary information within their component sets (experiments two and five), component sets not being able to consistently create structures due to rotational information (experiments three and four), and components creating structures consisting of at most two components (experiment one). Fisher's Exact Test (Cox and Snell 1989) (one-sided) for analysing binary data was used to analyse the results of the level two programming experiments (Table 6). The results are statistically significant, with a p value of 0. These successful results provide evidence to support our hypothesis at level two.

4.1.3 Level three: physical realisation of rule set for programming experiments

With the success of each system using a designed component set at level two, a level three translation was

Table 5 Designed and randomly generated component sets [represented as (Top, Left, Bottom, Right, Front, Back) × #, where the directions refer to component information locations and the # symbol represents quantity] for the programming experiments (PEX1–PEX5)

Experiment	Designed/random	Component set
PEX1	Designed	$(A_1,–,–,–,–,–) \times 9$, $(B_1,–,–,B_1,B_1,–) \times 3$
	Random	$(G_3,–,H_1,–,B_1,–) \times 9$, $(K_2,–,–,L_4,–,–) \times 3$
PEX2	Designed	$(C_1,–,–,–,–,–) \times 9$, $(E_1,–,–,–,D_1,–) \times 3$, $(F_2,D_1,–,D_1,–,–) \times 3$
	Random	$(–,A_1,–,C_1,C_1,–)$ x 9,$(–,–,–,C_1,–,G_3)$ x 3,$(F_2,–,–,L_2,J_4,–)$ x 3
PEX3	Designed	$(B_1,–,–,–,–,–) \times 6$, $(H_1,–,–,–,–,A_1) \times 3$, $(G_1,–,–,A_1,–,–) \times 3$
	Random	$(–,–,H_1,F_1,K_2,–) \times 6$, $(–,A_1,E_2,K_1,–,–) \times 3$, $(G_4,–,K_1,L_3,–,–) \times 3$
PEX4	Designed	$(D_1,–,–,–,–,–) \times 6$, $(I_1,–,–,–,C_1,–) \times 3$, $(J_1,–,–,–,C_1,–) \times 3$
	Random	$(B_1,–,–,–,–,I_4)$ x 6, $(–,–,–,H_1,J_1,E_2)$ x 3, $(G_4,–,–,A_1,–,F_1)$ x 3
PEX5	Designed	$(B_1,–,–,–,–,–)$ x 6, $(L_1,–,–,–,–,A_1)$ x 3, $(K_1,–,–,–,A_1,–)$ x 3
	Random	$(I_3,I_4,–,–,A_1,–) \times 6$, $(–,–,–,C_1,–,–) \times 3$, $(H_4,F_2,–,K_1,–,–) \times 3$

Table 6 The number of successful and unsuccessful trials for each designed and random programming experiment (PEX1–PEX5) at level two, with corresponding *p* value calculated using Fisher's Exact Test (one-sided) for analysing binary data

Experiment	Designed/ random	Successful	Unsuccessful	*p* value
PEX1	Designed	10	0	0
	Random	0	10	
PEX2	Designed	10	0	0
	Random	0	10	
PEX3	Designed	10	0	0
	Random	0	10	
PEX4	Designed	10	0	0
	Random	0	10	
PEX5	Designed	10	0	0
	Random	0	10	

performed to test if the translated component set of each experiment could self-assemble into its respective target structure. A level three translation was not performed on the systems using a randomly generated component set, since they were not successful. The level three experimental set-up and results are provided.

4.1.3.1 Level three: experimental set-up For each experiment trial, components were randomly placed in a jar of mineral oil, and placed on an orbital shaker (their environment). Details on the physical experimental set-up are provided in the Appendix. Each system was shaken for 20 min (similar to Bhalla et al. 2011), after which the state of each system was recorded, including observations for: the number of target structures created, the number of matching errors (between conflicting components), the number of rotation errors (between complementary components), and the number of assembly errors (partial attachment where no fits rule is applicable).

4.1.3.2 Level three: experimental results Each experiment was successful in creating at least one target structure (Fig. 9). Figure 10 shows an example of a successful trial from each experiment. In experiment one and two, there were no matching and rotation errors (as this was not possible due to the 5-magnetic-bit patterns present), and no assembly errors. Figure 11 shows the rotation errors for experiments three, four, and five. There were no matching and assembly errors in experiments three, four, and five. Fisher's Exact Test (one-sided) for analysing binary data was used to determine the statistical significance of creating target structures in each experiment (Table 7). All five programming experiments are statistically significant at the 0.01 level (i.e. there is a 99% percent certainty the results are not due to chance). These successful programming experiments provide evidence to support our hypothesis that given the attributes of a target structure, information encoded in the components could be used to direct the components to self-assemble into a three-dimensional target structure.

4.2 Evolving three-dimensional self-assembling systems

Our hypothesis for the evolutionary experiments was, given the attributes of a target structure, an evolutionary algorithm can be used to evolve a set of component rules, which can be mapped to a physical system consisting of an environment containing those components that are able to self-assemble into the three-dimensional target structure. The three-level approach with the incorporation of evolutionary computation was used to test our hypothesis. A

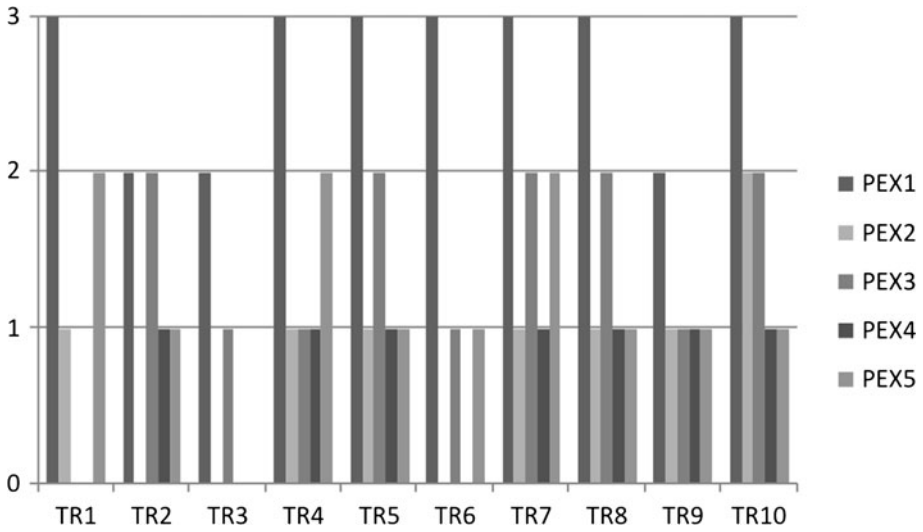


Fig. 9 Number of target structures created in each of the 10 trails (TR1–TR10), for each of the five programming experiments (PEX1–PEX5) at level three

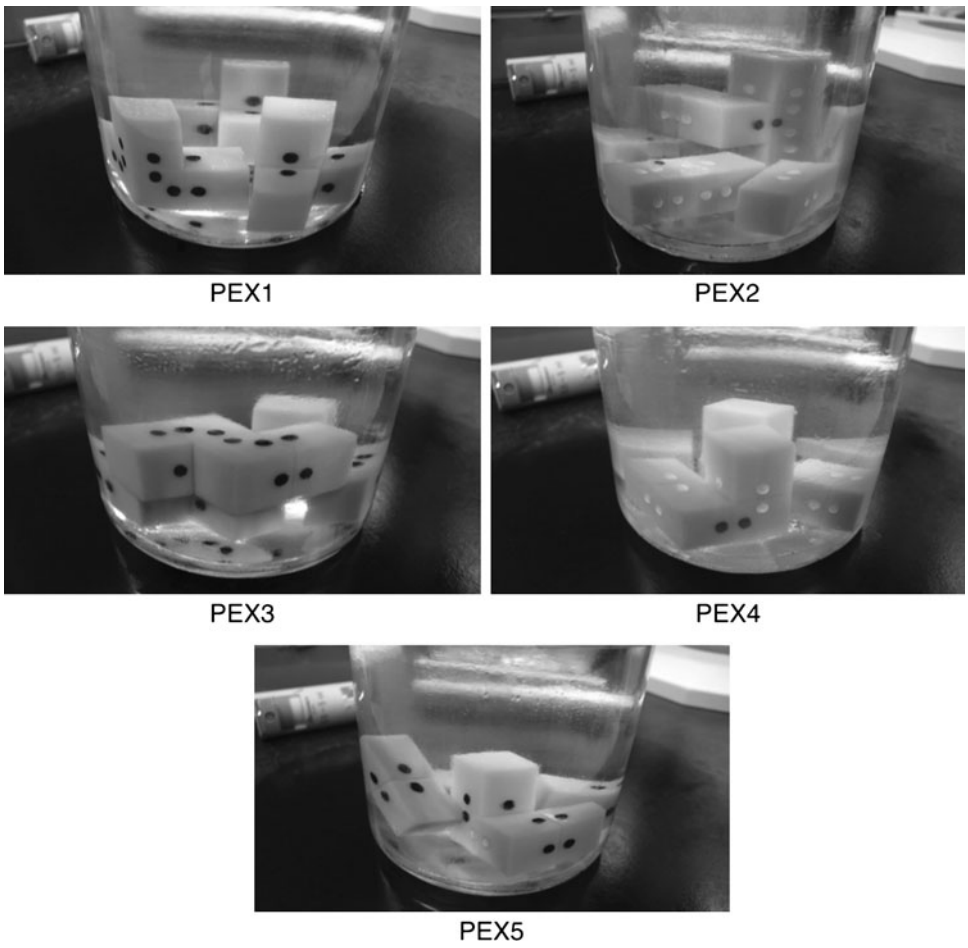


Fig. 10 Photographs of successful level three programming experiment trials (PEX1–PEX5)

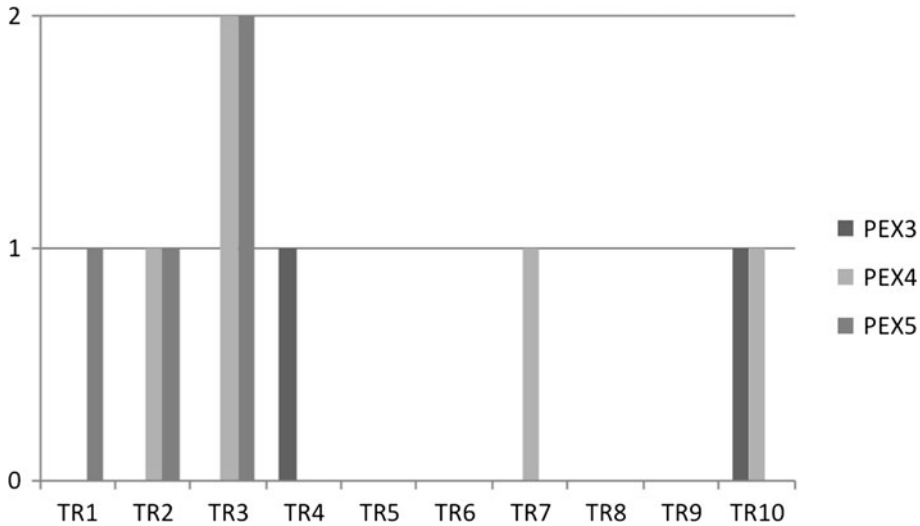


Fig. 11 Number of rotation errors in each of the 10 trails (TR1–TR10), for the last three programming experiments (PEX3–PEX5) at level three

target structure was assigned to each experiment (Fig. 8). As with the programming experiments, enough components were supplied to create up to three target structures, and 10 trials were run for each experiment. A trial was evaluated as successful if all three target structures were created for a virtual trial (at level two), and if at least one target structure was created for a physical trial (at level three). The three-level approach is used to describe the experimental procedure and results for the evolutionary experiments.

Table 7 Number of successful and unsuccessful trials for the programming experiments (PEX1–PEX5) at level three, with corresponding p values [calculated using Fisher's Exact Test (one-sided) for analysing binary data, with respect to 0 successful and 10 unsuccessful trials for the random sets for each programming experiment from their level two results]

Experiment	Successful	Unsuccessful	p value
PEX1	10	0	0
PEX2	7	3	0.002
PEX3	9	1	0
PEX4	7	3	0.002
PEX5	9	1	0

4.2.1 *Level one: definition of rule set for evolutionary experiments*

These three target structures were chosen for the same reasons as the target structures for the programming experiments. In these experiments, the independent variable is the set of components, which are defined by their type and concentration. In these experiments, the dependent variable is the resulting self-assembled structures. To test the independent variable, an evolved component set is generated along with a randomly generated component set.

The initial individual (genotype) length was set to the required number of components to create one target structure. Ten runs were conducted for each evolutionary experiment. For the randomly generated component set, components were created by selecting the information associated with each site with uniform probability. The number of components randomly generated were equal to the required number of components to create one target structure.

Although multiple solutions were evolved for each experiment, the selected solutions were based on those which used component designs previously used by the authors (in order to reduce the rapid prototyping financial costs). The number of components specified (evolved and random) were multiplied by three in order to increases the chances of being able to create up to three examples of the target structures. This method for creating the evolved and randomly generated component sets was chosen as part of our experimental set-up with the aim of providing proof-of-concept evidence to our evolutionary approach, by conducting a statistical significance test (details provided in the level two and three experimental results). The component sets used in the evolutionary experiments is provided in Table 8.

4.2.2 *Level two: virtual execution of rule set for evolutionary experiments*

3DcTAM was used to evaluate the ability of each self-assembly rule set (evolved or random) to create its respective target structures. Although the 3DcTAM is used by the evolutionary algorithm, it used to verify the creation of three target structures (Fig. 12). The level two experimental set-up and results are provided.

4.2.2.1 Level two: experimental set-up With the exception of component mapping following Table 8, the same level two experimental set-up from the programming experiments was used for the level two evolutionary experiments.

4.2.2.2 Level two: experimental results Table 9 provides the results of the evolutionary experiments at level two. Each evolved component set successfully created all three target structures, whereas the randomly generated components sets were unsuccessful in generating any target structures. Along with the level two programming experiments, these evolved results further support that seed tiles (components) are not required to create target structures. The causes for the unsuccessful randomly generated results follow the same reason as the level two programming experiments of component sets having uncomplementary information (experiments one and two), and component sets not being able to consistently create structures due to rotational information (experiment three). To analyse the

Table 8 Evolved and randomly generated component sets [represented as (Top, Left, Bottom, Right, Front, Back) × #, where the directions refer to component information locations and the # symbol represents quantity] for the evolutionary experiments (EEX1–EEX3)

Experiment	Evolved/random	Component set
EEX1	Evolved	(–,A_1,–,A_1,–,–) x 3, (–,–,B_1,–,–,–) x 6
	Random	(–,–,E_2,–,A_1,A_1) x 3, (D_1,H_1,–,–,–,–) x 6
EEX2	Evolved	(–,–,D_1,–,–,–) × 12, (–,–,–,E_2,C_1,C_1,) × 3, (C_1,–,C_1,–,–,F_1) × 3
	Random	(A_1,–,A_1,–,K_2,J_3) x 12, (–,K_4,D_1,–,–,–) x 3, (–,–,A_1,D_1,-,-) x 3
EEX3	Evolved	(–,–,–,–,–,B_1) × 9, (–,K_1,A_1,–,A_1,–) × 3, (A_1,–,–,–,L_4,–) × 3
	Random	(–,G_3,F_1,–,F_1,–,) × 9, (–,E_1,B_1,–,–,–) × 3, (–,D_1,–,–,–,E_2) × 3

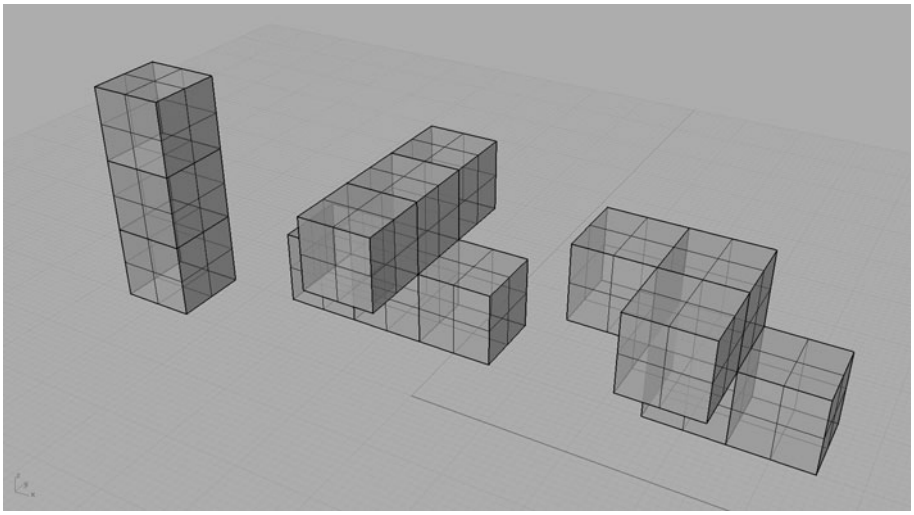


Fig. 12 Target structures for the evolutionary experiments (experiments 1–3 from *left to right*)

Table 9 The number of successful and unsuccessful trials for each evolved and random evolutionary experiment (EEX1–EEX3) at level two, with corresponding p value calculated using Fisher's Exact Test (one-sided) for analysing binary data

Experiment	Designed/ random	Successful	Unsuccessful	p value
EEX1	Designed	10	0	0
	Random	0	10	
EEX2	Designed	10	0	0
	Random	0	10	
EEX3	Designed	10	0	0
	Random	0	10	

evolutionary results, Fisher's Exact Test (one sided) for analysing binary data was used. These successful results provide evidence to support our hypothesis at level two.

4.2.3 *Level three: physical realisation of rule set for evolutionary experiments*

Each system using an evolved component set was successful at level two. As a result, a level three translation was performed to test if the translated component set of each evolved system could self-assemble into its respective target structure. Since the results were unsuccessful, a level three translation was not performed on the random systems. The level three experimental set-up and results are provided.

4.2.3.1 Level three: experimental set-up The same level three physical experimental set-up used in the programming experiments was used for the level three evolutionary experiments, with the exception of jar placements on the orbital shaker (see Appendix). At the conclusion of each physical evolutionary experiment trial, the state of system was recorded (number of target structures created, matching errors, rotation errors, and assembly errors).

4.2.3.2 Level three: experimental results Figure 13 shows the results for each evolutionary experiment at level three. Figure 14 shows an example of the final state of a successful trial, for each experiment. Rotation errors did not occur in experiment one and two, but did in experiment three (one rotation error in trial one). Matching and assembly errors did not occur in all three evolutionary experiments. Fisher's Exact Test (one-sided) for analysing binary data was used to determine the statistical significance of creating target structures in each experiment (Table 10). All three evolutionary experiments are statistically significant at the 0.01 level. These successful evolutionary experiments provide evidence to support our hypothesis that given the attributes of a target structure, an evolutionary algorithm could be used to evolve a set of component rules, which can be mapped to a physical system consisting of an environment containing components that are able to self-assemble into the three-dimensional target structure.

4.3 Discussion

Here we summarise the experimental results, consider areas of future work to improve our self-assembly design approach, and consider applications of our work.

4.3.1 *Experiment results summary*

Eight experiments were conducted to demonstrate two design methods using the three-level approach, programming (five experiments) and evolution (three experiments). At level two, all of the designed (programmed) and evolved component rule sets were successful in creating

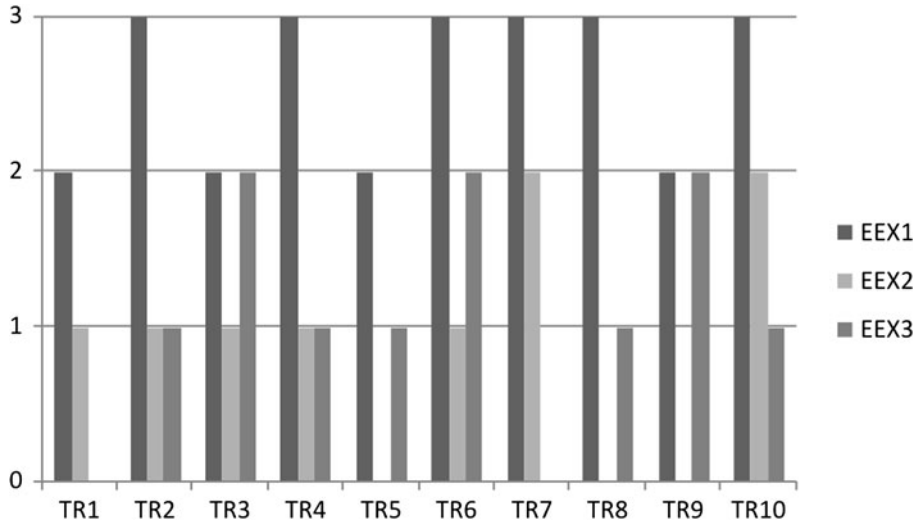


Fig. 13 Number of target structures created in each of the 10 trails (TR1–TR10), for each of the three evolutionary experiments (EEX1–EEX5) at level three

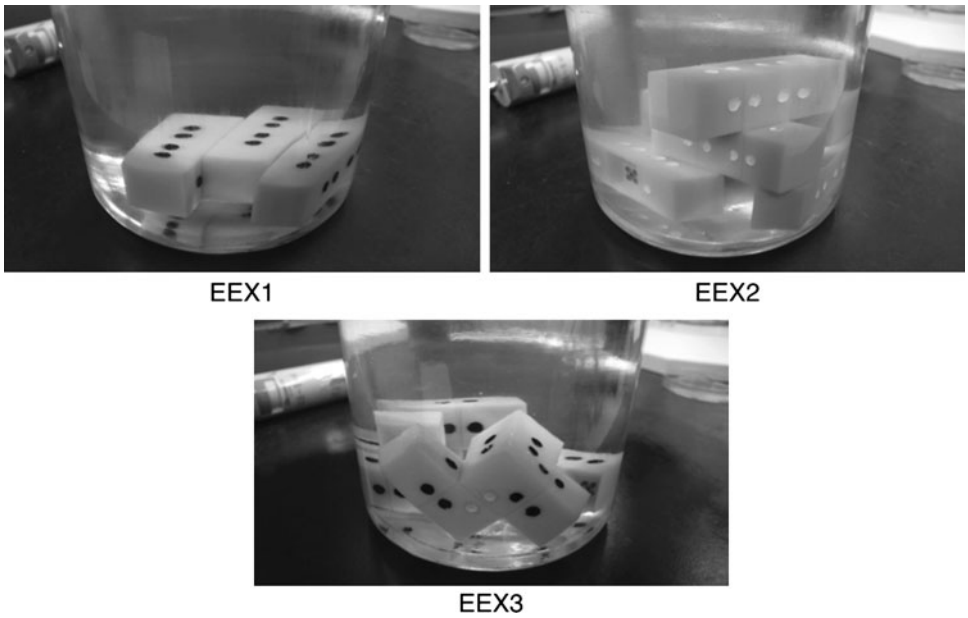


Fig. 14 Photographs of successful level three evolutionary experiment trials (EEX1–EEX3)

their respective target structures. In contrast, all of the level two randomly generated component sets (for both programming and evolutionary experiments) were unable to create a single respective target structure. At level three, all of the designed and evolved component rule sets were able to create at least one respective target structure. The level three physical results are at least statistically significant at the 1% level for both the programming and evolutionary experiments. These successful results add to those previously achieved by our three-level approach to self-assembly design (Bhalla et al. 2007, 2010, 2011; Bhalla and Bentley, in press). As well, these successful results support our proof-of-concept three-dimensional physical systems, including techniques to: encode component rotational information, reduce component-to-component error interactions, and direct the self-assembly process in three spatial dimensions.

4.3.2 Future work

To improve upon our proof-of-concept results, we consider a number of areas related to our self-assembly design approach as future work. These areas include improvements to the evolutionary algorithm used, simulation methods at level two, and extensions to the physical systems (considering both the environment and components).

Improvements to the evolutionary algorithm include conducting experiments to determine if there are better parameter settings for the number of generations, population size, and number of genotype evaluations, as well as for the

Table 10 Number of successful and unsuccessful trials for the evolutionary experiments (EEX1–EEX5) at level three, with corresponding p values [calculated using Fisher's Exact Test (one-sided) for analysing binary data, with respect to 0 successful and 10 unsuccessful trials for the random sets for each evolutionary experiment from their level two results]

Experiment	Successful	Unsuccessful	p value
EEX1	10	0	0
EEX2	7	3	0.002
EEX3	8	2	0

selection, crossover, and genetic operators. Similarly, investigations into objective weightings, and new objectives (e.g. when considering refined solutions), should also be conducted to improve multi-objective fitness evaluations. Lastly, enhancements to the canonical genetic algorithm used here should also be considered, as another method to improve the evolved results. As future work we look to being able to evolve multiple target structures simultaneously.

An overall improvement to the three-level approach would be augmentations of the level two modelling. Such an augmentation includes the use of both an abstract, tile-based model (e.g. 3DcTAM) for computationally efficient evaluation of a set of self-assembly rules, along with a physics-based model for evaluation of a translated set of self-assembly rules. For example, only self-assembly rule sets that are successful in the tile-based model would be evaluated using the physics-based model for a more efficient use of computational resources. Such an augmentation would be beneficial when evolving self-assembling systems, as a better assessment of the complexity of the evolved structures while minimising the number of physical evaluations could be conducted.

We are also considering extensions to the physical systems, to both the environment and components, for more robust results. Although it was qualitatively observed that structures would assemble, disassemble, and new structures would emerge periodically, a better experimental set-up to record quantitative environment observations would assist in making improvements to the environment. New experiments for testing the various environment variables (e.g. shaking speed and duration) would also lead to more robust physical results. Evaluating the resulting environment data would be required to also consider future experiments where changes in the environment (such as fluctuations in temperature) could be used to direct the self-assembly process (referred to as *temperature programming*; Kao and Schweller 2006). As well, we look to further investigate self-assembling systems in three dimensions featuring higher complexity, using components with more sophisticated morphologies (including shape spaces and higher-order magnetic-bit patterns).

4.3.3 Applications

In general, the advantages of three-dimensional self-assembly include the creation of structures that make a more efficient use of volume, as well as shorter interconnections between components (Whitesides and Gryzbowski 2002). Specifically, we envision our three-level approach being applicable to the design of structures (including at the nano and microscale), circuit fabrication, modular and swarm robotics control systems, synthetic biology, and DNA computing using self-assembly. For example, three-dimensional DNA computing has been proposed, but the required components have not existed at the time of writing (Pelletier and Weimerskrich 2002). Furthermore, potential applications for three-dimensional self-assembly includes the creation of hybrid rapid prototyping technologies that make use of self-assembly at larger physical scales (Hiller and Lipson 2009), and potentially at the nanoscale (Gates et al. 2005).

5 Conclusions

We presented an extension of our three-level approach to address the shortcomings of designing and creating self-assembling systems in three spatial dimensions. The work presented here progresses techniques to solve an open problem in self-assembly, of being able to create a set of components and their environment, such that the components self-assemble into a target structure. The contributions of this work includes the development of new self-assembly rules that include rotational information, the creation of the 3DcTAM which is Turing Universal at temperature one, and the demonstration of new physical components to create target structures with symmetric and asymmetric features in three dimensions. We presented eight proof-of-concept experiments to demonstrate design methods for three-dimensional self-assembly. The successful results demonstrate how the three-level approach, by incorporating evolutionary computation, can be used for programming and evolving physical self-assembling systems in three dimensions.

Acknowledgments We would like to thank the two anonymous reviewers for their thoughtful and insightful comments.

Appendix

The following is a list of the materials and methods for constructing physical components and their corresponding environment, as well as the physical experimental procedure (at level three, using the three level approach) for the programming and evolution experiments.

Component materials and methods

The materials, including tools, used for constructing components include:

- Eden 333 Polyjet rapid prototyping machine
- Vero Gray resin
- Neodymium (NdFeB) disc magnets; 1/16″ × 1/32″ (diameter × height), grade N50
- Magnetic pole identifier
- Magnet placement tools
- Vice; built from a PanaVise bench clamp mount, a PanaVise low profile base, and a PanaVise low profile head
- Sharpie paint pens; oil based, fine point (colours: black, white, purple, yellow, green, and orange) and extra fine point (colours: red and blue)
- Rhino3D version 4.0 computer-aided design (CAD) software

The method for constructing the components has the following eight steps:

1. Create the CAD files using Rhino3D for the magnet placement tool (specifications provided in Fig. 15) and for the components, based on the specifications in Fig. 16 and in association with the components' shape space (to determine key, lock, and neutral information locations on each component, as well as any component interaction markers on neutral shapes adjacent to information locations).
2. Fabricate four magnet placement tools (two with a magnet in the centre, and two with a magnet in the corner) and the components using an Eden 333 Polyjet rapid prototyping machine with Vero Gray resin and the CAD files.
3. Insert three neodymium disc magnets in each magnet placement tool, to create one tool with magnetic north polarity and the other with magnetic south polarity (identify polarity using the magnetic pole identifier).
4. Paint the magnet placement tools, using the sharpie paint pens (blue for magnetic north and red for magnetic south), to complete the construction of the magnet placement tools.
5. Insert magnets into the components by first identifying the appropriate 5-magnetic-bit pattern for each key and lock, and then placing magnets on the appropriate magnet placement tool (two magnets for a key shape and three magnets for a lock shape) and using the vice to insert the magnets into the appropriate location in the 5-magnetic bit pattern (Fig. 17).
6. Remove the magnet placement tool and component from the vice and separate the component placement tool and the component (the extra magnet will dislodge and create an air gap on the component where the magnets were inserted; this follows the component physical encoding scheme for minimising key-to-lock error interactions provided in Table 3).
7. Repeat steps seven and eight until all magnets have been inserted into all the components.
8. Paint, using the sharpie paint pens, the magnets that have been placed in the components (blue for magnetic north and red for magnetic south) and any component interaction markers (refer to Fig. 4 to use the appropriate colours), to complete the construction of the components.

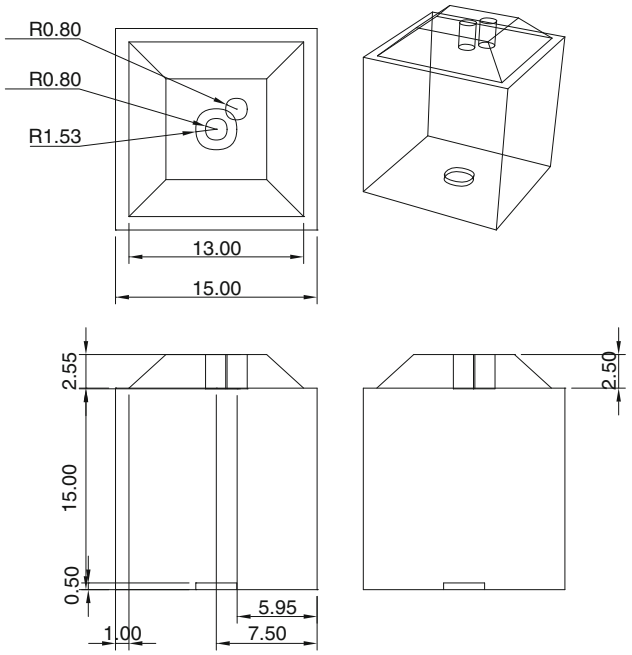


Fig. 15 Magnet placement tool specifications (counter-clockwise from *top right*: perspective, top, front, right views), with all construction units in mm; note two magnet locations are shown at the top of the component (centre and corner), however only one magnet location should be used per tool (two locations are show here to reduce the number of technical drawings)

Environment materials and methods

The materials, including tools, used for constructing the environment include:

- Trotec Speedy 300 Laser Engraver laser cutting machine
- Acrylic sheet; transparent, 3 mm in height
- Mill board; 0.8 mm in height
- Screws; first type (5/8″ in length, pan head, 6-32 UTS, 18-8 grade stainless steel), second type (1/4″ in length, pan head, 6-32 UTS, 18-8 grade stainless steel)
- Hex nuts; 6-32 UTS, 18-8 grade stainless steel

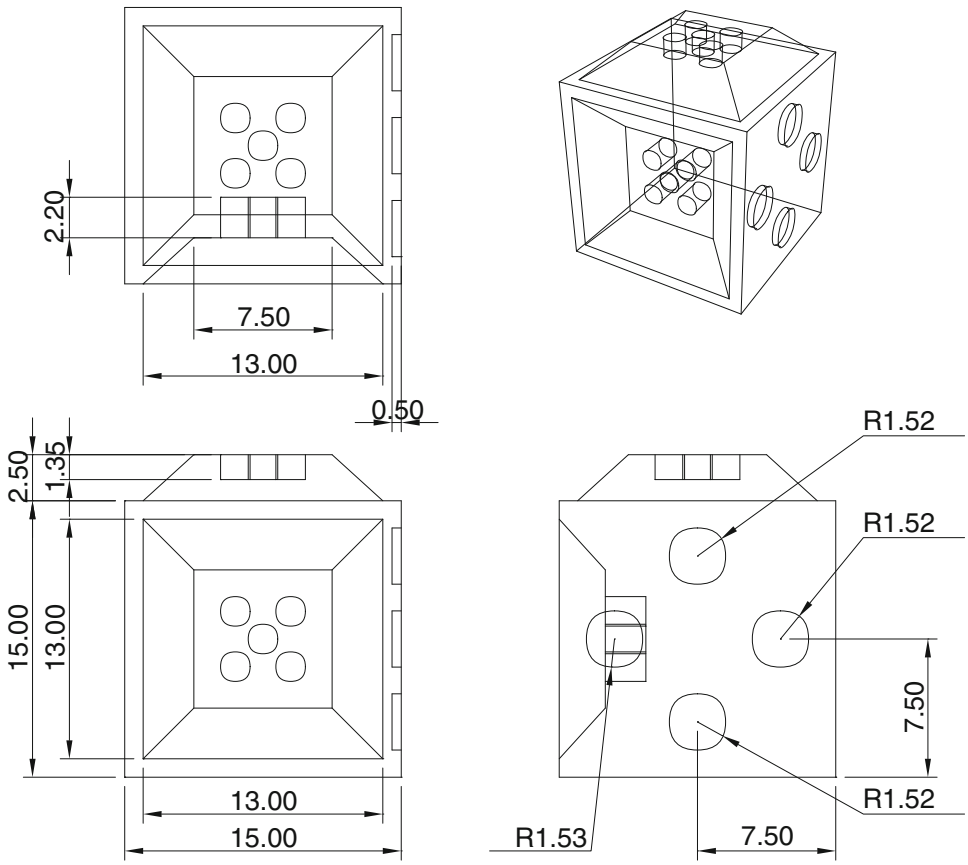


Fig. 16 Component specifications showing the base component dimensions, key shape, lock shape, and interaction markers (counter-clockwise from *top right*: perspective, top, front, right views), with all construction units in mm

- Glue; Loctite, regular, gel
- Angle brackets; 1″, stainless steel
- New Brunswick Scientific Excella E1 Platform Shaker
- Adobe Illustrator Creative Suite 4

The method for constructing the environment has the following six steps:

1. Create the CAD files using Adobe Illustrator for the jar rack parts (Fig. 18).
2. Fabricate the jar rack parts (two bottom parts a and b, two side parts, and one top part) using a Trotec Speedy 300 Laser Engraver laser cutting machine using mill board for the jar sleeve and acrylic for the remaining environment jar rack parts, and the CAD files.
3. Glue the three bottom jar rack parts together (bottom parts a both below bottom part b).
4. Construct the jar rack by using the screws, hex nuts, and corner braces to secure the bottom, sides, and top parts of the jar rack, and using the screws and hex nuts to secure the jar sleeve to the top of the jar rack.
5. Fold the overhang pieces of the jar rack sleeve over the holes for the jars in the top par of the jar rack.
6. Place the jar rack on the shaker (Fig. 19), and secure the jar rack to the shaker using the screw on the side of shaker's platform (these screws are supplied with the shaker), to complete construction of the environment.

Physical experimental procedure

The materials required to conduct the physical experimental procedure include:

- Graduated cylinder
- Mineral oil; Rogier Pharma light grade
- Stopwatch
- Jars; VWR clear glass wide mouth, plastic lid with rubber liner, 500 ml capacity, 91 mm by 95 mm (diameter by height)

Fig. 17 Example of constructing a component during a vice press, and magnet placement tool (*left*) and component (*right*) inside the vice

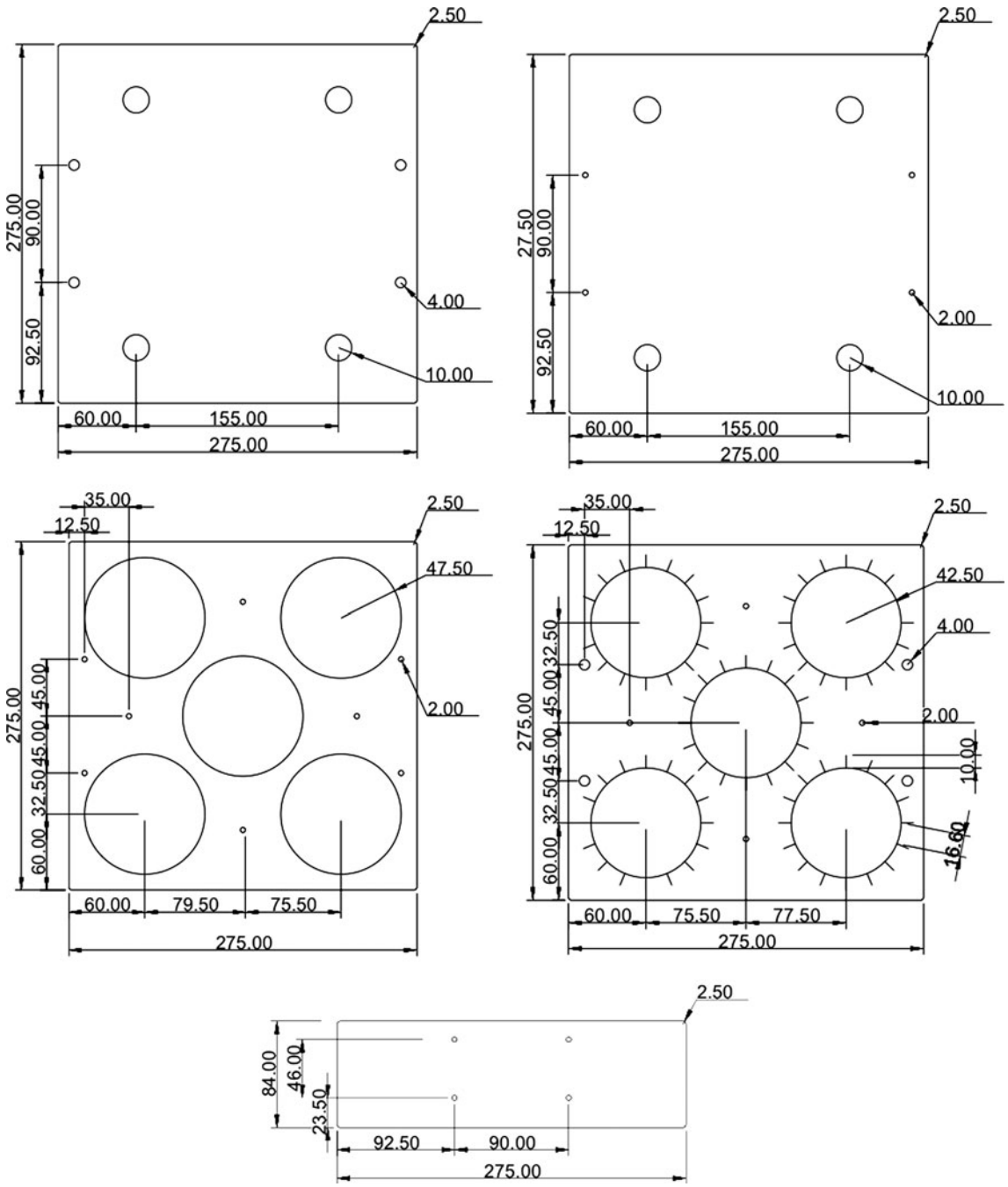


Fig. 18 Environment parts for jar rack (from *top left to bottom*: bottom part A, bottom part B, top part A, jar sleeve, and side part), with all construction units in mm

The physical experimental procedure, for both the programming and evolutionary experiments, has the following seven steps:

1. Create the CAD files using Adobe Illustrator for the jar rack parts (Fig. 19, specifications provided in Fig. 18).
2. Measure 325 ml of mineral oil using the graduate cylinder for each jar used in the programming experiments (five jars) and the evolutionary experiments (3 jars).
3. Place the jars of mineral oil in the jar rack (Fig. 20).
4. Randomly place the components for each experiment into the appropriate jar, and secure the jar lid.
5. Turn the shaker on by setting the speed to 32.5 rotations per minute, and start the stopwatch.
6. Stop the shaker after 20 min.
7. Record the state of each system.

Fig. 19 Photograph of the environment with the jar rack secured to the shaker, and a jar with mineral oil containing components as an example (jars of mineral oil are used to finalise the component's environment; details are provided in physical experimental procedure)

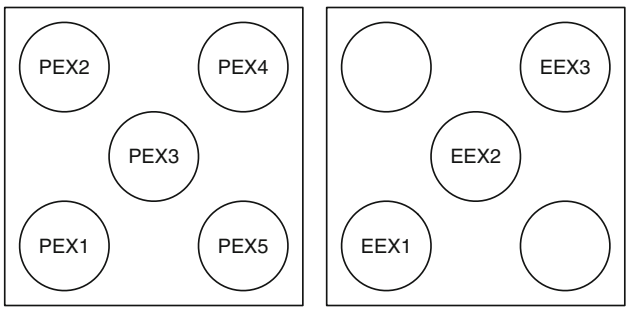


Fig. 20 Jar configuration (*top view*) for the programming experiments (*left*, PEX1–PEX5) and the evolutionary experiments (*right*, EEX1–EEX3), where the *labels* is the figure correspond to the experiment numbers provided in Tables 5 and 8, respectively

References

Adleman L (1994) Molecular computation of solutions to combinatorial problems. Science 266(5187):1021–1024

Adleman L, Cheng Q, Goel A, Huang M-D, Kempe D, de Espanés PM, Rothemund PWK (2002) Combinatorial optimization problems in self-assembly. In: 34th ACM international symposium on theory of computing, New York

Ampatzis C, Tuci E, Tuci V, Christensen AL, Dorigo M (2009) Evolving self-assembly in autonomous homogeneous robots: experiments with two physical robots. Artif Life 15(4):1–20

Ball P (1999) The self-made tapestry pattern formation in nature. Oxford University Press, Oxford

Bentley PJ, Wakefield JP (1997) Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. In: Chawdry PK, Roy R, Pant RK (eds) Soft computing in engineering design and manufacturing. Springer Verlag London Limited, Heidelberg, pp 231–240

Berger R (1966) Memoirs of the american mathematical soceity—the undecidability of the domino problem. American Mathematical Society, Providence

Bhalla N, Bentley, PJ. Programming physical self-assembling systems via physically encoded information. In: Doursat R, Sayama H, and Michel O (eds) Morphogenetic engineering: toward programmable complex systems. NECSI "Studies on Complexity" Series. Springer-Verlag (in press)

Bhalla N, Bentley PJ, Jacob C (2007) Mapping virtual self-assembly rules to physical systems. In: Proceedings of the international conference on unconventional computing (UC 2007), Bristol

Bhalla N, Bentley PJ, Jacob C (2010) Evolving physical self-assembling systems in two-dimensions. In: Proceedings of the international conference on evolvable systems (ICES 2010)

Bhalla N, Bentley PJ, Vize PD, Jacob C (2011) Staging the self-assembly process using morphological information. In: Proceedings of European conference on artificial life, Paris, France

Blasquez I, Poiraudeau J-F (2003) Efficient processing of Minkowski functionals on a 3D binary image using binary decision diagrams. J WSCG 11(1)

Boncheva M, Bruzewicz DA, Whitesides GM (2003a) Formation of chiral, three-dimensional aggregates by self-assembly of helical components. Langmuir 19:6066–6071

Boncheva M, Bruzewicz DA, Whitesides GM (2003b) Millimeter-scale self-assembly and its applications. Pure Appl Chem 75(5):621–630

Boncheva M, Andreev SA, Mahadevan L, Winkleman A, Reichman DR, Prentiss MG, Whitesides S, Whitesides GM (2005) Magnetic self-assembly of three-dimensional surfaces from planar sheets. Proc Natl Acad Sci 102(11):3924–3929

Brun Y (2008a) Constant-size tileset for solving an NP-complete problem in nondeterministic linear time. In: Garzon MH, Yan H (eds) DNA. Springer, Berlin, pp 26–35

Brun Y (2008b) Solving NP-complete problems in the tile assembly model. Theor Comput Sci 395:31–46

Buchanan A, Gazzola G, Bedau MA (2008) Evolutionary design of a model of self-assembling chemical structures. In: Krasnogor N, Gustafson S, Pelta DA, Verdegay JL (eds) Systems self-assembly: multidisciplinary snapshots. Elsevier, Amsterdam

Callicott N (2001) Computer-aided manufacture in architecture: the pursuit of novelty. Architectural Press, Oxford

Chen J, Seeman NC (1991) Synthesis from DNA of a molecule with the connectivity of a cube. Nature 350:631–633

Cook SA (1971) The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on theory of compuitng (STOC), pp 151–158

Cook M, Fu Y, Schweller R (2011) Temperature 1 self-assembly: deterministic assembly in 3D and probabilistic assembly in 2D. In: Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms (SODA 2011)

Corne D, Knowles J (2007) Techniques for highly multiobjective optimisation: some nondominated points are better than others. In: Lipson H (ed) Genetic and evolutionary computing conference, London, pp 773–780

Cox DR, Snell EJ (1989) Analysis of binary data. Chapman and Hall/CRC Press, London

Crick FHC (1970) Central dogma of molecular biology. Nature 227:561–563

Culik K (1996) An aperiodic set of 13 wang tiles. Discr Math 160:245–251

Douglas SM, Dietz H, Liedl T, Högberg B, Graf F, Shih WM (2009) Self-assembly of DNA into nanoscale three-dimensional shapes. Nature 459:414–418

Garey M, Johnson DS (1979) Computers and intractability: a guide to the theory of np-completeness. W.H. Freeman and Company, San Francisco

Gates BD, Xu Q, Stewart M, Ryan D, Wilson CG, Whitesides GM (2005) New approaches to nanofabrication: molding, printing, and other techniques. Chem Rev 105:1171–1196

Gracias DH, Tien J, Breen TL, Hsu C, Whitesides GM (2000) Forming electrical networks in three dimensions by self-assembly. Science 289(5482):1170–1172

Gracias DH, Kavthekar V, Love JC, Paul KE, Whitesides GM (2002) Fabrication of micrometer-scale, patterned polyhedra by self-assembly. Adv Mater 14(3):235–238

Groß R, Dorigo M (2008) Self-assembly at the macroscopic scale. Proc IEEE 96(9):1490–1508

Hiller J, Lipson H (2009) Design and analysis of digital materials for physical 3D voxel printing. Rapid Prototyp J 15(2):137–149

Jonoska N, McColm GL (2009) Complexity classes for self-assembling flexible tiles. Theor Comput Sci 410:332–346

Jr ERJ, Eisenberg E, Mazurek D (2009) Vector mechanics for engineers: statics. McGraw-Hill Higher Education, New York

Kao M-Y, Schweller R (2006) Reducing tile complexity for self-assembly through temperature programming. In: Proceedings of the 7th annual ACM-SIAM symposium series on discrete algorithms, Miami, FL, pp 571–580

Kari L, Mahalingam K (2010) Watson–crick palindromes in DNA computing. Nat Comput 9:297–316

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum Press, New York, pp 85–103

Li L, Siepmann P, Smaldon J, Terrazas G, Krasnogor N (2008) Automated self-assembling programming. In: Krasnogor N, Gustafson S, Pelta DA, Verdegay JL (eds) Systems self-assembly: multidisciplinary snapshots. Studies in multidisciplinary, vol 5. Elsevier, Amsertdam, pp 281–307

Michielsen K, de Raedt H (2000) Morphological image analysis. Comput Phys Commun 132:94–103

Mitchell M (1996) An introduction to genetic algorithms. MIT Press, Cambridge

Murphy MP, O'Neill LAJ (1997) What is life?: the next fifty years: speculations on the future of biology. Cambridge University Press, Cambridge

Păun G, Rozenberg G, Salomaa A (1998) DNA computing—new computing paradigms. Springer, New York

Pelesko J (2007) Self assembly: the science of things that put themselves together. Chapman and Hall/CRC Press, Boca Raton

Pelletier O, Weimerskrich A (2002) Algorithmic self-assembly of DNA tiles and its applications to cryptanalysis. In: Proceedings of the international conference on genetic and evolutionary computation (GECCO 2002)

Rothmund PWK (2005) Design of DNA origami. In: Proceedings of the international conference on computer-aided design (ICCAD 2005)

Rothmund PWK (2006) Folding DNA to create nanoscale shapes and patterns. Nature 440:297–302

Rothmund PWK, Winfree E (2000) The program size complexity of self-assembled squares. In: ACM symposium on theory of computing (STOC 2000)

Schrödinger E (1944, reprinted 2003) What is life? with mind and matter and autobiographical sketches. Cambridge University Press, Cambridge

Seeman NC (1999) DNA engineering and its application to nanotechnology. Trends Biotechnol 17(11):437–443

Seeman NC (2004) Nanotechnology and the double helix. Sci Am 290(6):64–75

Seeman NC (2007) An overview of structural DNA nanotechnology. Mol Biotechnol 3(37):246–257

Sipser M (1997) Introduction to the theory of computation. PWS Publishing Company, Boston

Soille P (2003) Morphological image analysis. Springer, Berlin

Stepney S, Braunstein SL, Clark JA, Tyrrell T, Adamatzky A, Smith RE, Addis T, Johnson C, Timmis J, Welch P, Milner R, Partridge D (2005) Journeys in non-classical computation I: a grand challenge. Int J Parallel Emerg Distrib Syst 20(1):5–19

Stepney S, Braunstein SL, Clark JA, Tyrrell T, Adamatzky A, Smith RE, Addis T, Johnson C, Timmis J, Welch P, Milner R, Partridge D (2006) Journeys in non-classical computation II: initial journeys and waypoints. Int J Parallel Emerg Distrib Syst 21(2):97–125

Terfort A, Bowden N, Whitesides GM (1997) Three-dimensional self-assembly of millimeter-scale components. Nature 386:162–164

Terrazas G, Gheorghe M, Kendall G, Kranogor N (2007) Evolving tiles for automated self-assembly design. In: Proceedings of the 2007 IEEE congress on evolutionary computation (CEC2007) 2001–2008

Thompson, D. W. (1942, reprinted 1992). On growth and form. In: Bonner JT (abridged ed). Cambridge University Press, Cambridge

Wang H (1961) Proving theorems by pattern recognition. Bell Syst Tech J 40(1):1–41

Wang H (1965) Games, logic and computers. Sci Am 213:98–106

Watson JD, Crick FHC (1953) Molecular structure of nucleic acids—a structure for deoxyribose nucleic acid. Nature 171(4356): 737–738

Whitesides GM, Boncheva M (2002) Beyond molecules: self-assembly of mesoscopic and macroscopic components. PNAS 99(8):4769–4774

Whitesides GM, Boncheva M (2005) Making things by self-assembly. MRES Bull 30:736–742

Whitesides GM, Gryzbowski G (2002) Self-assembly at all scales. Science 295:2418–2421

Winfree E (1995) On the computational power of DNA annealing and ligation. DNA Based Comput 27:199–221

Winfree E (1998a) Simulations of computing by self-assembly. In: Proceedings of 4th international meeting on DNA based computers

Winfree E (1998b) Algorithmic self-assembly of DNA. PhD dissertation, California Institute of Technology, Pasadena

Winfree E (1999) Algorithmic self-assembly of DNA: theoretical motivations and 2D assembly experiments. J Biomol Struct Dyn 11(2):263–270

Winfree E, Liu F, Wenzier LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394(6): 539–544

Wolfram S (2002) A new kind of science. Wolfram Media, Champaign

Wu H, Thalladi VR, Whitesides S, Whitesides GM (2002) Using hierarchical self-assembly to form three-dimensional lattices of spheres. J Am Chem Soc 124(48):14495–14502

Zhang Y, Seeman NC (1994) Construction of a DNA-truncated octahedron. J Am Chem Soc 5(116):1661–1669

Zykov V, Mytilinaios E, Adams B, Lipson H (2005) Self-reproducing machines. Nature 435(7038):163–164