Design and modular self-assembly of nanostructures

Joakim Bohlin

Balliol College
University of Oxford

A thesis submitted for the degree of Doctor of Philosophy

Michaelmas 2021

Abstract

As nucleic acid nanostructures grow larger and more complex, new tools and methods are needed to facilitate their design. DNA origami structures, for example, are limited by the lengths of their scaffolds, but can they be joined together into larger multi-component designs. This thesis presents two projects, each approaching the design of such modular structures at a different level of abstraction.

Project 1 presents the *polycube* self-assembly model, where building blocks assemble stochastically using complementary patches. The assembly of both 2D polyominoes, as well as 3D polycubes, is considered. First, the mapping between input rules (defining the set of available species) and output shapes is investigated, revealing a clear bias toward low-complexity structures. The frequent shapes also tend to be highly modular and symmetric. Secondly, the reversed mapping is explored, presenting a method to find the minimal rule that assembles a provided output shape.

Project 2 presents a more detailed approach to the design of modular structures and individual modules; the *oxView* toolkit for the design, analysis, and visualisation of DNA, RNA and protein nanostructures. While many other design tools exist, oxView makes it easy to import and connect their designs into complete assemblies. Furthermore, oxView allows for free-form editing and rigid-body manipulation. Designs can then be interactively simulated using the oxDNA model, providing a more intuitive understanding of the resulting dynamics.

In conclusion, nanostructures self-assembly design has been investigated on both an abstract and a more detailed level. The presented projects have resulted in tools and methods for creating, simulating and analysing self-limiting modular structures with minimal complexity, potentially containing building blocks created in multiple design softwares.

Design and modular self-assembly of nanostructures

Joakim Bohlin

Balliol College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2021

This thesis is dedicated to
Ellen ♡ʖ
for joining me on this adventure

Acknowledgements

First and foremost, I would like to thank my supervisors, Professor Andrew Turberfield and Professor Ard Louis, for giving me this opportunity to begin with, and for all their help, knowledge, and support. Andrew has provided sound and rational advice whenever I have needed it, helping me to ensure the scientific relevance of my work and making sure I focus on what is important. Likewise, Ard has provided much appreciated advice, inspiration, and ideas for new research directions and collaborations.

I would also like to thank my fellow members of the Turberfield lab. For their help, for a friendly working environment and for excellent beta testing of my creations; Dr. Jonathan Bath, Rafael Carrascosa Marzo, Dr. Erik Benson, Dr. Seham Helmi, Dr. Antonio Garcia Guerra, Behnam Najafi, Dr. Emma Silvester, Dr. Robert Oppenheimer, Catherine Fan, Alma Chapet–Batlle, Sing Ming Chan, Qian Zhang, and Dr. Rana Abdul Razzak.

I am also very thankful to Professor Petr Šulc, at Arizona State University, for his advice and guidance concerning oxView development, patchy particle simulation, as well as oxRNA. The secondment I spent in the Šulc group was a very valuable and productive time. My fellow oxView developers Erik Poppleton, Michael Matthies, Jonah Procyk, and Aatmik Mallya also deserve gratitude for their hard and excellent work.

Similarly, I would also like to thank Prof. Ebbe Andersen at Aarhus University for my secondment in his group and for introducing me to RNA origami. Also, thank you Néstor Sampedro for all the help and valuable discussions.

Finally, I also appreciate the help and input from Prof. Jonathan Doye and his group, including Hannah Fowler, Dr. Domen Prešern and others.

On a more personal note, for her undying support (despite my endless ramblings about polycubes and simulations), my beloved wife Ellen Bohlin simply cannot be thanked enough. Thank you for joining me on yet another adventure and for enduring the times I still had to leave you behind.

Of course, also want to thank the rest of my family. My father Ulf, who I am certain would also have loved to study here had he been given the opportunity. My mother Okki, for her creativity, her care, and encouragement. My two sisters, Ann-Marie and Ingela, for inspiring me to travel and to study.

I am likewise grateful for the help and encouragement I have received from Ellen's side of the family. My parents-in-law Karl-Johan and Ann-Louise are like an extra set of parents in their support, but I also want to specifically acknowledge Ellen's late grandfather, Anders Bohlin. A botanist in the spirit of Linnaeus, he kept on learning and teaching until the very end.

Our time in Wheatley would not have been the same without such wonderful neighbours; thank you Bruce, Helmer, Harriet and Peter, for making us feel at home and for all the fun we managed to have despite the pandemic.

Finally, I also want to thank our taido and karate friends for keeping us sane and in relative shape during these exciting years: Gunnar and (once again) Rafa for Balliol taido, Emily and Asami for welcoming us to the Exeter club, Jesper and Malin for visiting us in Oxford and through zoom. Also, everyone at the Wheatley Ryobu-kai karate club for many fun sessions.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 765703

Abstract

As nucleic acid nanostructures grow larger and more complex, new tools and methods are needed to facilitate their design. DNA origami structures, for example, are limited by the lengths of their scaffolds, but can they be joined together into larger multi-component designs. This thesis presents two projects, each approaching the design of such modular structures at a different level of abstraction.

Project 1 presents the *polycube* self-assembly model, where building blocks assemble stochastically using complementary patches. The assembly of both 2D polyominoes, as well as 3D polycubes, is considered. First, the mapping between input rules (defining the set of available species) and output shapes is investigated, revealing a clear bias toward low-complexity structures. The frequent shapes also tend to be highly modular and symmetric. Secondly, the reversed mapping is explored, presenting a method to find the minimal rule that assembles a provided output shape.

Project 2 presents a more detailed approach to the design of modular structures and individual modules; the *oxView* toolkit for the design, analysis, and visualisation of DNA, RNA and protein nanostructures. While many other design tools exist, oxView makes it easy to import and connect their designs into complete assemblies. Furthermore, oxView allows for free-form editing and rigid-body manipulation. Designs can then be interactively simulated using the oxDNA model, providing a more intuitive understanding of the resulting dynamics.

In conclusion, nanostructures self-assembly design has been investigated on both an abstract and a more detailed level. The presented projects have resulted in tools and methods for creating, simulating and analysing self-limiting modular structures with minimal complexity, potentially containing building blocks created in multiple design softwares.

Contents

List of Figures

List of Tables

Glossary

2D, 3D . . . . Two- or three-dimensional, referring in this thesis to spatial dimensions of a self-assembly structure.

AIT . . . . . Algorithmic Information Theory

DNA . . . . . Deoxyribonucleic acid.

RNA . . . . . Ribonucleic acid.

PDB . . . . . The Protein Data Bank. Used in this thesis to refer to the file format used to save atomic structures.

Polycube . . . A 3D shape consisting of multiple cubes connected by their sides.

Polyomino . . A 2D shape consisting of multiple squares connected by their edges.

SAT . . . . . Boolean satisfiability.

Species . . . . Used in this thesis to denote a type of cube allowed by the polycube rule.

TEM . . . . . Transmission Electron Microscopy

Far out in the uncharted backwaters of the unfashionable end of the western spiral arm of the Galaxy lies a small unregarded yellow sun. Orbiting this at a distance of roughly ninety-two million miles is an utterly insignificant little blue green planet whose ape-descended life forms are so amazingly primitive that they still think digital watches are a pretty neat idea.

— D. Adams, The Hitchhiker's Guide to the Galaxy

1 Introduction

Contents

How do you design and assemble something on the nanoscale? When building something on the macroscale, it is easy to pick the building blocks you want and use your hands to attach them where you want them to be. However, for nanoscale objects, it is difficult to have that level of top-down control. Instead, more success has been had by imitating nature and letting the building blocks assemble themselves. After all, with millions of years of evolution, nature has quite the advantage. This thesis will cover novel methods and tools to design such self-assembling nanostructures.

1.1 Thesis structure

The thesis covers two related projects, both concerning the design and modular self-assembly of nanostructures. Each project is introduced by a separate chapter on its relevant history and background.

The first project, introduced by a background in Chapter 2, covers an abstract self-assembly model called *polycubes*. Chapter 3 details the model and the shapes we get when randomly sampling the input space. Chapter 4 presents the results on the inverse problem; given a polycube shape, what input rules will assemble it and what is the least complex input you can find?

The second project takes a more detailed view of self-assembly design, with Chapter 5 providing a background on computer-aided design tools for nucleic acid structures, together with some related simulation models. Chapter 6 then presents my contributions to *oxView*, a web-based tool for the visualisation, design, and integration of DNA, RNA and protein structures.

Finally, Chapter 7 provides some concluding remarks, with a discussion on the results of both projects.

Before moving on to the individual projects, however, the following sections will provide a general background on nucleic acid self-assembly and complexity.

1.2 DNA design

The main building material covered in this thesis is DNA. An abbreviation of deoxyribonucleic acid, DNA is a string-like molecule more known for encoding the genes of living systems [1]. These strands of DNA are made up of units called *nucleotides*, consisting of a sugar-phosphate backbone as well as one of four possible bases: *adenine* (**A**), *thymine* (**T**) *cytosine* (**C**), and *guanine* (**G**).

Through Watson-Crick base-pairing, the **A** nucleotide form two hydrogen bonds with **T**, while **G** forms three with **C**, making DNA double-stranded. Each strand has a directionality, conventionally represented as going from the 3' to the 5' end of the strand, making the duplex *anti-parallel*.

The base part of the nucleotide is *hydrophobic*, while the sugar-phosphate backbone is *hydrophilic*, which means that the bases "hide" on the inside of the duplex to avoid contact with water molecules. However, the backbone distance is about 6 Å (0.6 nm), while the bases would need to be at a distance of 3.3 Å (the thickness of the base) to not leave any room for water [1]. To solve this inequality,

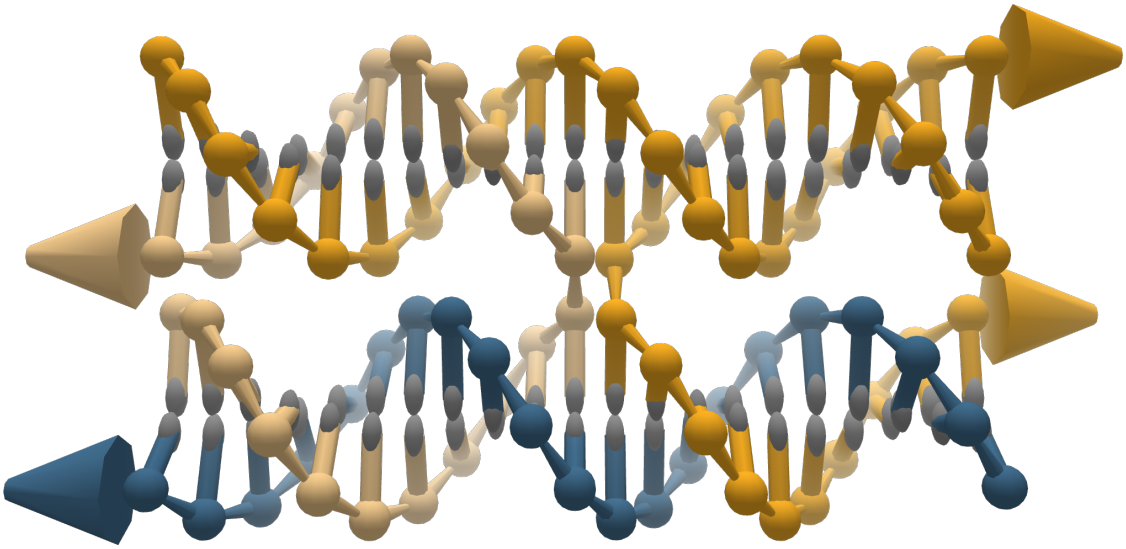

Figure 1.1: Holliday junction, designed in oxView. Cones at the end indicates the 3' end of each of the four strands.

the DNA duplex forms a double-helix structure with a radius of about 9 Å, placing everything at an energetically comfortable distance.

While a single double-helix is the most natural confirmation, it is possible for strands to branch into multiple junctions. For example, the Holliday junction is a junction between four double-helical arms, shown in Figure 1.1 in one of its possible configurations.

By designing sequences with complementary domains for the intended duplex regions, it is possible to create many different DNA motifs and structures [2, 3], an understanding that pioneered the field of structural DNA nanotechnology and the use of DNA as a building material.

Another breakthrough in the field was the DNA origami technique [4] a now popular and proven method for creating larger irregular structures using DNA. The principle behind it, as illustrated in Figure 1.2, is to use short staple strands to fold one long viral scaffold strand into the desired structure. Using design software, it has become relatively easy to design structures of any given form. See Section 5.1 for an introduction to such tools.

In DNA origami, the size of the structure is limited by the length of the scaffold, which motivates researchers to investigate modular approaches. Some previous

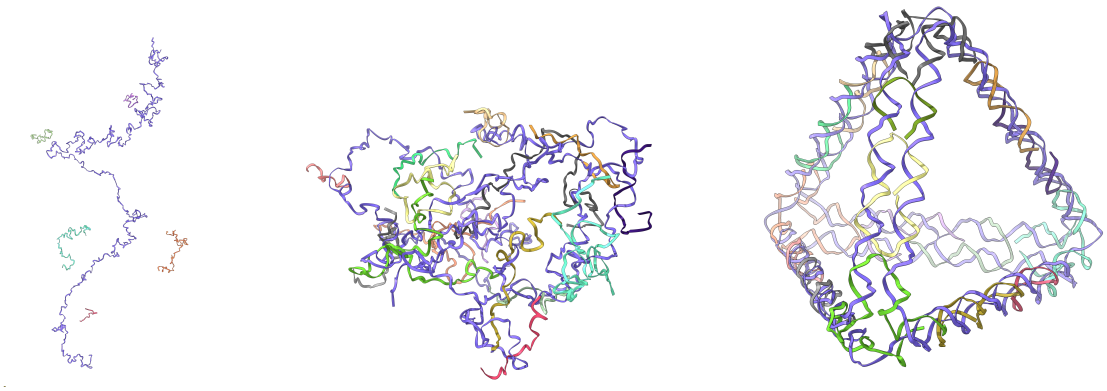

Figure 1.2: Illustration of DNA origami self-assembly of a tetrahedron. A long scaffold strand (purple), obtained from a virus, is folded into the desired shape by multiple short staple strands binding to complementary domains of the scaffold. Tetrahedron design obtained from `https://cando-dna-origami.org/examples/` and melted using oxDNA simulation [5].

results will be described in Section 2.1, but the results of this thesis aim to simplify the design process significantly.

1.3 RNA design

Another promising building material for self-assembly is ribonucleic acid (RNA). RNA is very similar to DNA, but with the *thymine* base (**T**) replaced by *uracil* (**U**).

Biologically, DNA is transcribed into RNA by the RNA polymerase enzyme as part of gene expression. While DNA folding is easier to predict, the fact that RNA is more reactive than DNA also offers the possibility of a more useful structure; for example, by incorporating aptamers, enzymes, and other such functionalities [6].

In 2014, Geary et al., from the Andersen lab in Aarhus, demonstrated a method [7–9] for co-transcriptionally folded RNA origami, which also enables folding *in vivo*. As shown in Figure 1.3, the design used a set of tertiary RNA motifs (such as kissing hairpins and double crossovers) as modules. These modules are then combined into a single-stranded blueprint, for which a sequence that should co-transcriptionally fold into the intended shape is found.

The Andersen lab is one of the partners to the ITN network I am part of, and I have spent a two-month secondment there working with their RNA origami method. Some of my results on simulating RNA designs are covered in Section 6.6.

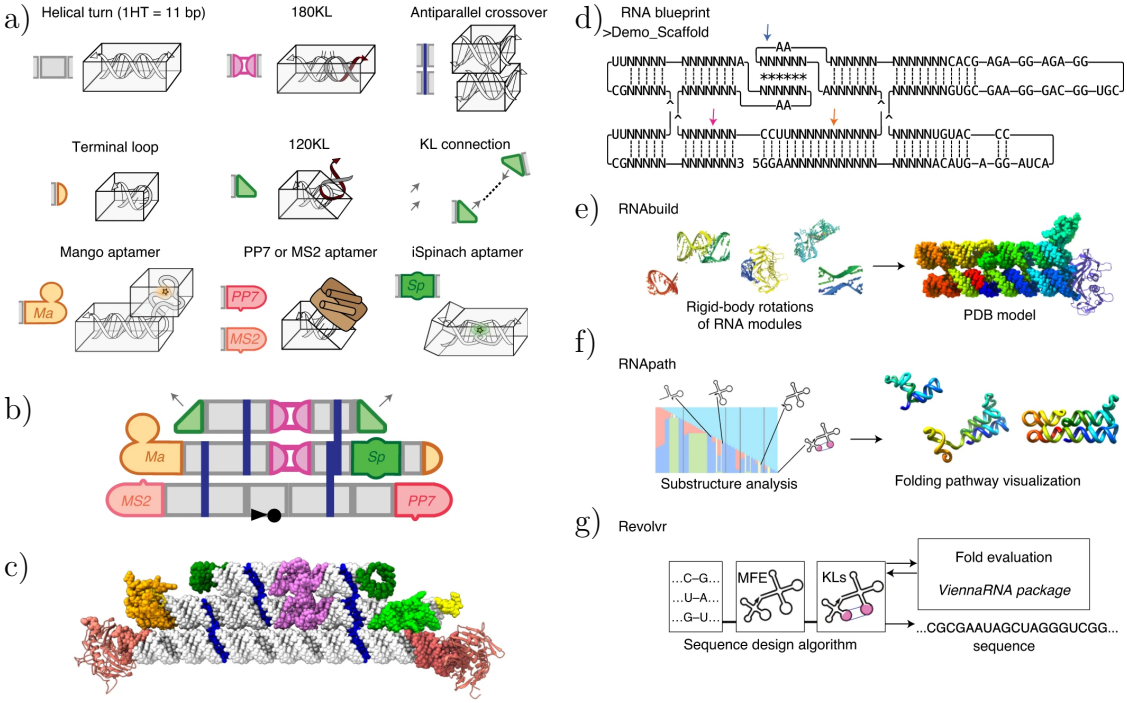

Figure 1.3: Co-transcriptional folding of RNA origami, adapted from [9]. **a)** The set of RNA motifs used as modular building blocks. **b)** Schematic of the modules connected to form a single strand. **c)** Atomistic model of the design in b). **d)** Shows the text-based blueprint used to create designs, while **e)**, **f)**, and **g)** shows scripts developed to aid the visualisation and preform sequence design for the origami.

1.4 Algorithmic Information Theory and input-output maps

Besides possible self-assembly materials, another important concept in this thesis is the notion of *complexity*. Complexity has different definitions in different fields, but here we focus on the amount of information needed to describe something. Some things, whether in the form of a shape, a song, or a binary string, require less information to describe than others, and we would then call those less complex than their counterparts.

Let us consider the complexity of text strings. If you have a monkey pressing random keys on a typewriter, you would expect it to produce every string of length N with equal probability (assuming the keystrokes were indeed truly random). With k keys on the keyboard, the probability for any string of length N is then k^{-N}. For example, the title of this thesis, while unlikely to appear randomly, would

be equally as probable as any other 50-character string, see the three example strings below, all with probability k^{-50}:

```
1  DESIGN AND MODULAR SELF–ASSEMBLY OF NANOSTRUCTURES
2  SHWDRVWKFORWJDOEXOZLSBNREKC Z  VSDJJF  ROKFYRVMIUI
3  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

But what if we described our strings using algorithms, rather than a full listing of the letters it contains? For example, string number three above could then be described using the C programming language as the algorithm:

```
printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
```

But a much shorter description can be found as follows:

```
for(int i=50; i--;) printf("A");
```

There are also a number of possible valid variations of the code above, with different variable names and coding conventions, all producing the same output. In other words, not only is the description shorter than writing out the full string but multiple inputs map to the same output, increasing the probability of that particular output further. It also feels intuitive that a string repeating a single letter should be less complex than a random-looking alternative.

This concept of using the shortest possible computer program that can describe an object (for example, a binary string) to determine its complexity is central within the field of Algorithmic Information Theory (AIT) and is called *Kolmogorov complexity* (or Solomonoff–Kolmogorov–Chaitin to give full credit) [10]. More specifically, the Kolmogorov complexity $K(x)$ of an output x is the length of the shortest program that generates x on a Universal Turing Machine (UTM).

AIT also includes the *coding theorem*, introducing lower and upper bounds for the probability $P(x)$ of generating a binary string x as $2^{-K(x)} \leq P(x) \leq 2^{-K(x)+\mathcal{O}(1)}$. In other words, low-complexity outputs are exponentially more likely to be generated by random input compared to high-complexity outputs. This could be compared to how

there are many more programs generating the "simple" string number three above compared to the randomly generated string two or the carefully selected string one.

However, a problem with Komologrov complexity is that finding the shortest program for a given output is far from trivial (it is, in fact, *uncomputable* in general, due to the *halting problem*). Fortunately, Dingle at al [11] were able to derive an upper bound to the probability using a computable approximation $\widetilde{K}(x)$ of the Komologrov complexity:

$$P(x) \lesssim 2^{-a\widetilde{K}(x)-b}$$

where a and b are constants that depend on the input-output map used (but are independent of x). This is very helpful for calculating the complexity of self-assembled shapes, where properties of the simplest known input rule can be used as such a Komologrov complexity proxy, as will be seen in Section 2.2.3 and Chapter 3.

2

An introduction to modular assembly

Contents

From the start, the field of structural DNA nanotechnology has been interested in modular assemblies, although initially in the form of infinite and uniform crystal structures. In fact, Professor Ned Seeman was inspired to pioneer the field after seeing the woodcut *Depth* by M.C. Escher, where fish are depicted organised into a neat crystalline structure [2].

However, components that self-assemble into chrystals can also assemble bounded structures, if only a way can be found to make them self-limiting. The question

is how many unique components would be required, relating to the complexity question introduced in Section 1.4.

This chapter will provide an overview of the background of self-assembled multi-component structures, both bounded and unbounded, starting with experimental results of ever-increasing size and followed by a selection of theoretical models.

2.1 Experimental applications

From small tiles made from a handful of strands, to megadalton-scale structures made from multiple origami designs, modular self-assembly has seen considerable experimental research. This section provides a quick overview to some results of particular interest to the polycube model presented in Chapter 3.

2.1.1 DNA tiles

Following early nucleic acid multi-arm junctions and lattices suggested by Seeman [12], Winfree [13, 14] used double-crossover (DX) motifs, shown in Figure 2.1.a, to self-assemble 2D DNA crystals. As can be seen in figure 2.1.b, the lattices could be made with varying complexity, exemplified using either two or four different species of tiles.

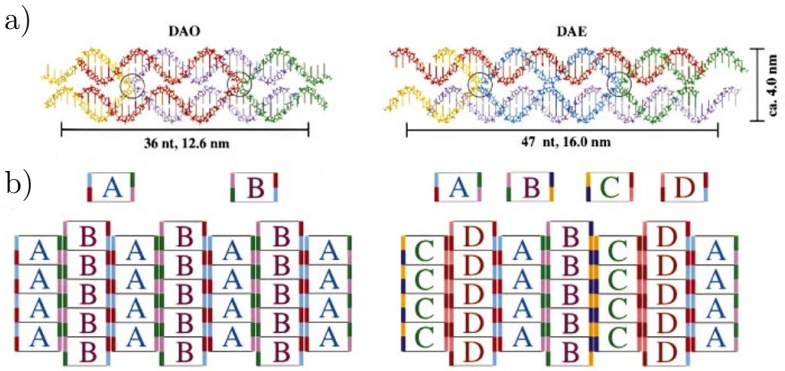

Figure 2.1: DX tiles forming 2D lattices, adapted from [14]. **a)** Examples of tile designs with double-crossover motifs. **b)** Lattices made using two and four tile species respectively.

2.1.2 DNA bricks

A three-dimensional DNA "canvas" was created in 2017 by Ong et al. [15], called *DNA bricks*. Structures are assembled from up to about 30,000 unique components, as seen in Figure 2.2. Since each component consists of a unique strand, custom shapes can be "sculpted" by leaving out the voxels (3D pixels) not required.

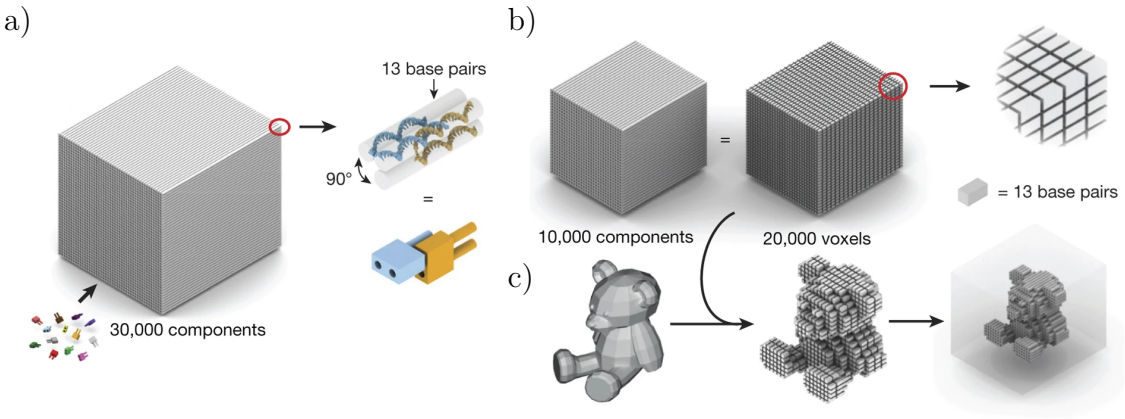

Figure 2.2: DNA bricks, adapted from [15]. **a)** DNA brick structure, where each of the up to 30'000 unique components is a 52 nucleotide DNA strand. The strands connect through a 13 base pair complementary domain at a 90 degree dihedral angle. **b)** A cuboid, here shown with 10,000 components, corresponds to a 20,000 voxel canvas. **c)** Approximating the shape of a teddy bear by removing a subset of the voxels from the canvas.

2.1.3 RNA tiles

As already covered in Section 1.3, it is possible to co-transcriptionally fold *RNA origami* [7]. This was first shown in 2014 by Geary et al., who folded RNA tiles that connect through complementary 120-degree kissing loop interactions, as seen in Figure 2.3, forming a hexagonal lattice. A significant promise with co-transcriptionally folded RNA structures is that they can be assembled *in vivo*.

2.1.4 Finite DNA origami arrays

In 2017, Tikhomirov et al.[16, 17] used the DNA origami technique to demonstrate two-dimensional patterns assembled on the micrometre-scale using square tiles where each tile was a complete origami, as seen in Figure 2.4. The tiles connect to each other through complementary single-stranded overhangs on their edges.

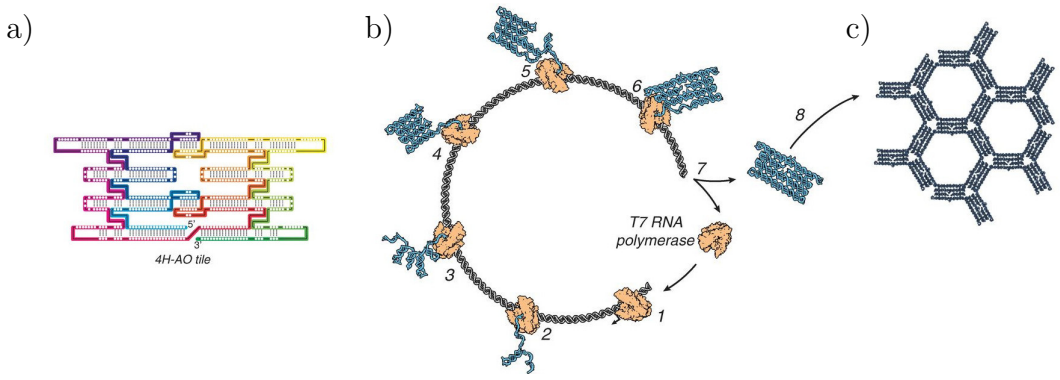

Figure 2.3: Co-transcriptional folding RNA origami tiles, adapted from [7]. The tiles connect through 120-degree kissing loop interactions, forming a hexagonal lattice. **a)** Detailed scematic of the four-helix 4H-AO tile. **b)** Co-transcriptional folding, where the RNA tile folds as it is transcribed from a DNA template. **c)** Hexagonal lattice formed by folded tiles.

The patterns could either be hierarchically assembled from unique tiles [16], or assembled into random patterns from a small number of tile types [17]. The binding strength could be calibrated using a variable amount of edge overhangs, as seen in Figure 2.4.b). Arrays up to 8×8 tiles were successfully produced, although larger arrays had a much smaller yield (Figure 2.4.c).

While the random tilings were generally unbounded, Tikhomirov et al. also showed how to program a finite grid, as seen in Figure 2.4.d). These tiles are very similar to the polyomino model described in Section 2.2.3.

2.1.5 Shape-complementary origami

Also, in 2017, Wagenbauer et al. [18] used shape-complementarity to assemble DNA origami components into three-dimensional polyhedral shapes up to 450 nanometers in diameter. Later, in 2021, Sigl et al. assembled large shells from shape-complementary origami triangles, as seen in Figure 2.5. The triangular sides attach through helix protrusions and indentations of complementary shape, as seen in figure 2.5.a), binding together through blunt-end stacking interactions, a method first shown by Woo et al. [19].

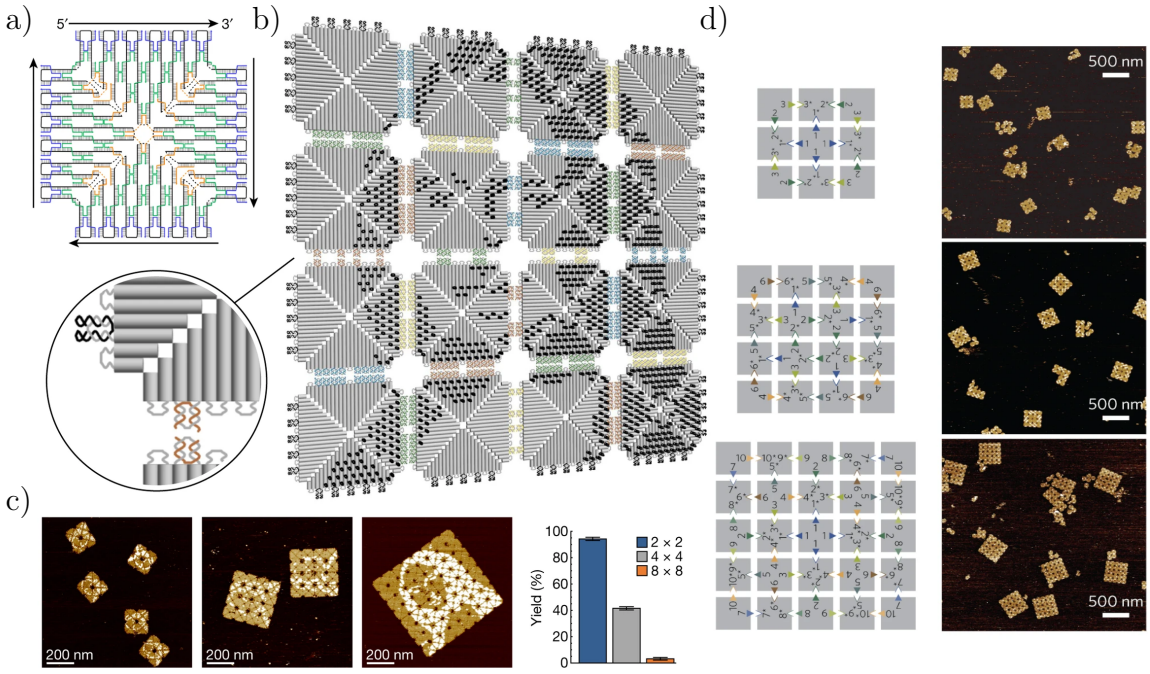

Figure 2.4: DNA origami arrays, adapted from [16, 17]. **a)** Strand-level diagram of a 12×12 version of the the origami tile (actual size is 22×22 helices). **b)** 4×4 tile "Mona Lisa" pattern. **c)** AFM image of patterned assemblies of different sizes (left) with their respective yields (right). **d)** Abstract design diagrams (left) and AFM images (right) of finite origami arrays, designed to different sizes [17].

2.1.6 DNA origami nanochambers

In 2020, Lin et al.[21] presented cubic DNA origami "nanochambers". The chambers have sticky-end overhangs on every side of the cube, allowing it to assemble in one, two and three dimensions, as can be seen in Figure 2.6. However, since the shape is only rotationally symmetric around the z-axis, the assembly is a still only a limited subset of the polycube model described in Chapter 3.

2.1.7 Octahedral DNA origami frames

In 2020, Wang et al. [22] showed how octahedral DNA origami frames could be used as building blocks in limited and unlimited programmed assemblies, as seen in Figure 2.7. Using sticky-end overhangs at the octahedral vertices, the building blocks have the connectivity, as well as the rotational symmetry, of a cube.

Due to the flexibility of the single-stranded connections, the connections are less torsionally rigid than assumed in the polycube model, later described in

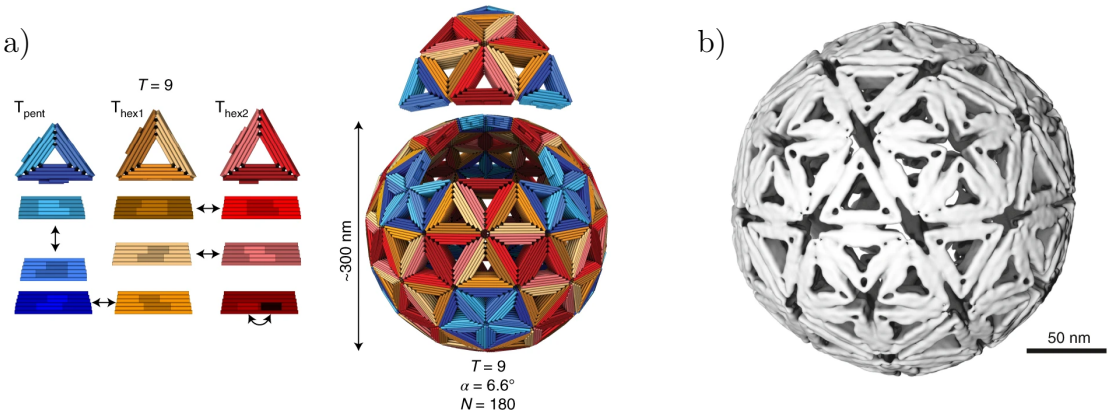

Figure 2.5: Shape-complementary triangles assembling polyhedral shells. Adapted from [20]. **a)** Polyhedral shell design for T=9. N is the triangulation number (the number of unique edges required for assembly), α is the bevel angle of the triangle sides, and N is the number of triangles required for a full shell. **b)** Cryo-EM reconstruction of an assembled T=4 icosahedral shell.

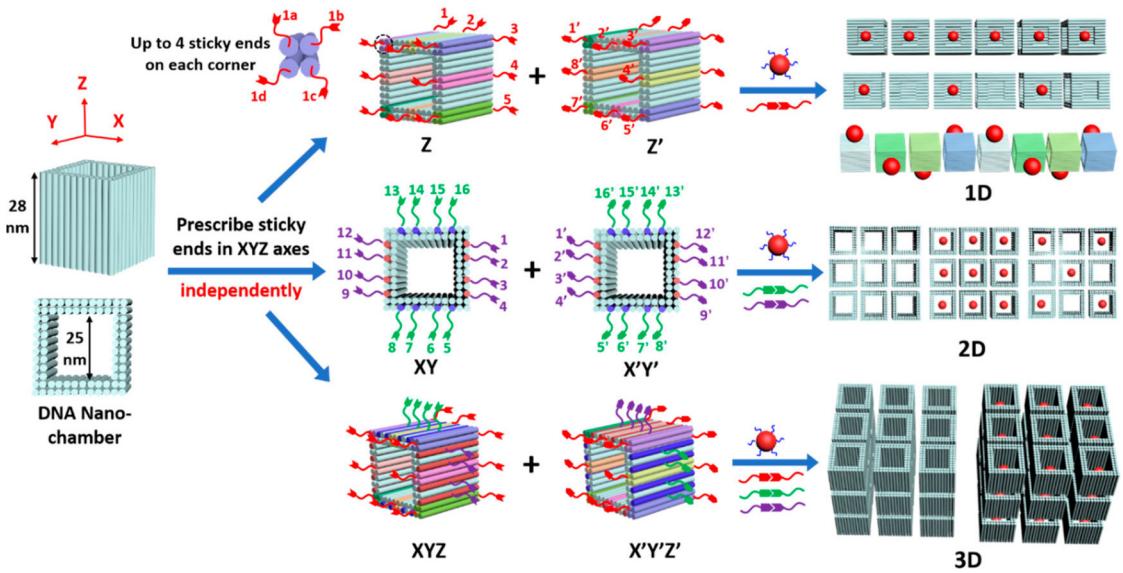

Figure 2.6: DNA origami nanochambers, adapted from [21]. **a)** Concept illustration, where building blocks with *polychromatic* bonds (differentiated though different single-stranded sequences), assemble into 1D, 2D, and 3D structures. **b)** Schematic of DNA nanochamber programmable assembly, showing sticky end overhangs applied in 1D, 2D, and 3D assemblies.

Chapter 3. However, as can be seen in Figure 2.7, shapes such as the cube can still be assembled with good yield.

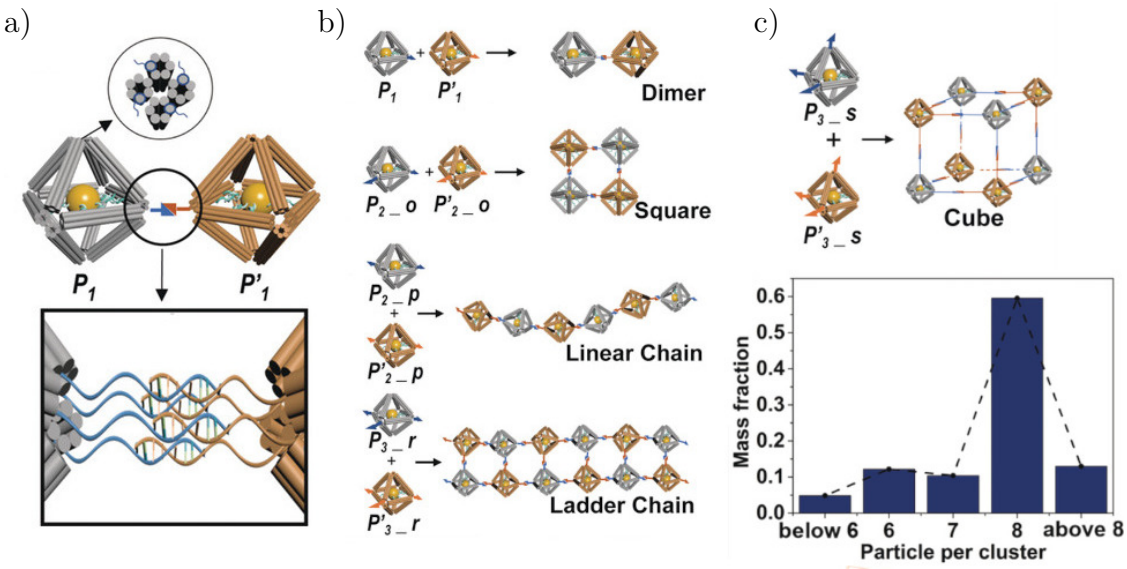

Figure 2.7: Octahedral DNA origami frames, adapted from [22]. **a)** Two octahedra with complementary sticky ends binding together to form a dimer. The edges consist of six-helix bundles. **b)** nanoclusters assembled from different sets of building blocks. **c)** $2 \times 2 \times 2$ cube nano cluster (top) and histogram of the mass fraction, where the intended design of eight components per cluster is the most common.

2.2 Modular assembly models

It would be computationally unreasonable to simulate module assembly only at the level of individual nucleotides. A simpler approach is to use an abstract model to predict the assembly process with the modules treated as rigid bodies or discrete tiles. This section will present earlier such models as a background to my own polycube model which will be introduced in Chapter 3.

2.2.1 Wang tiles

Introduced by Hao Wang in 1961 [23], so-called *Wang tiles* are square tiles with a colour on each of their four edges. Without rotating or reflecting the tiles, they assemble so that adjacent edges have the same colour.

The DNA tiles by Winfree et al. [14], presented in Section 2.1.1, behave like Wang tiles by design and do not allow rotations or reflections. Winfree investigated the possibility of using such tiles for computation [13], which led to aTAM: the algorithmic Tile Assembly Model.

2.2.2 The algorithmic tile assembly model

The algorithmic Tile Assembly Model (aTAM), shown in Figure 2.8, models the dynamic behaviour of the double crossover DNA tiles introduced by Winfree et al. [14]. Each tile has four patches, one on each edge, corresponding to the four single-stranded regions of the DNA tile. Furthermore, the patches can have different strengths, with a global temperature variable determining the total connection strength required for a tile to attach [24].

In the example seen in Figure 2.8, the pattern grows from the initial bottom-right seed into the blue horizontal bottom row and the rightmost vertical column. This is because these tiles have "strength-2" glues with enough binding strength to attach by themselves [24]. The additional tiles have weaker "strength-1" glues (illustrated as thinner black connectors), so they need at least two complementary patches to achieve the binding strength threshold (temperature).

Because of this so-called *co-operative binding*, the tiles can be seen as logic gates performing computation; given the bottom and right patches as input bits, the matching tile attaches and produces two computed output bits (top and left).

2.2.3 The polyomino model

The main inspiration for the *polycube* model (presented in Chapter 3) is the polyomino model [26, 27]. As noted in Section 2.1.4, the 2D model is similar in assembly to the later experimental micro-meter scale tile designs by Tikhomirov [17] shown in Figure 2.4.d). Compared to the aTAM model described in Section 2.2.2, polyomino tiles are allowed to rotate, creating further possibilities for symmetries. Also, the edge binding is not self-complementary, with complementary colour pairs used instead. Finally, polyominoes have a constant binding strength (temperature-1), assembling without co-operative binding.

See Figure 2.9 for an illustration of the model, where an input *genotype* genotype (describing the four possible tile types) is mapped into an assembled output polyomino by stochastically growing the shape from an initial seed. The growth stops if, as in the figure, no more tiles can attach (since the colour 0 does

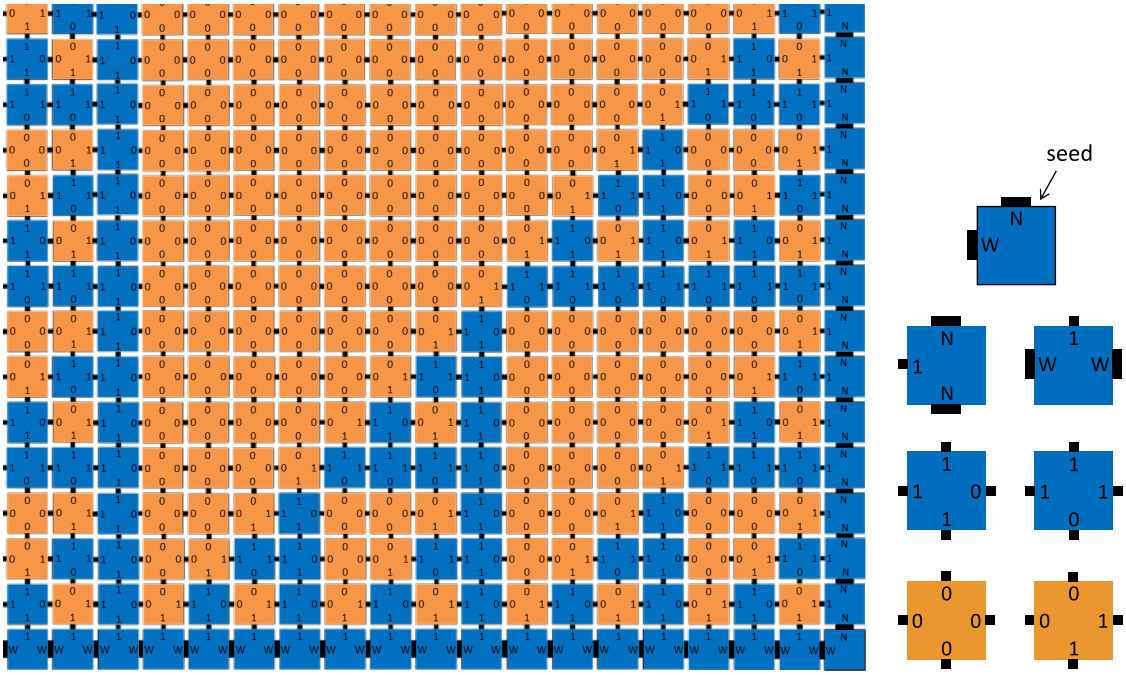

Figure 2.8: Algorithmic self-assembly of a Sierpiński triangle. Adapted from [25]. A tile set (right) grows from an initial seed by co-operatively attaching self-complementary edges (without rotation). The 0 and 0 "glues" are weaker and require two matching bonds to attach (co-operative binding), compared to the W (west) and N (north) glues that are strong enough to bind alone.

not bind to anything). If the growth is infinite, the genotype is called *unbounded*. A genotype is considered *deterministic* if it assembles the same phenotype polyomino every time. Only bounded and deterministic genotypes are considered valid inputs.

Evolutionary runs of polyominoes showed a clear bias toward structures with low complexity and high symmetry [28]. See Figure 2.10, where the frequency of protein complexes and polyominoes are compared to their complexity. To be clear, the number of interface types required (number of patch colours in the polyomino case) is used as a proxy measure for the Komologrov complexity of the structure (see Section 1.4).

The evolutionary fitness of the polyominoes only depended on their size (16-mers had the highest fitness), so the simplicity was not selected for but is instead a property of the mapping. This finding is in line with the Algorithmic Information Theory arguments made by Dingle et al. [11], who showed the same simplicity bias for a set of various input-output maps.

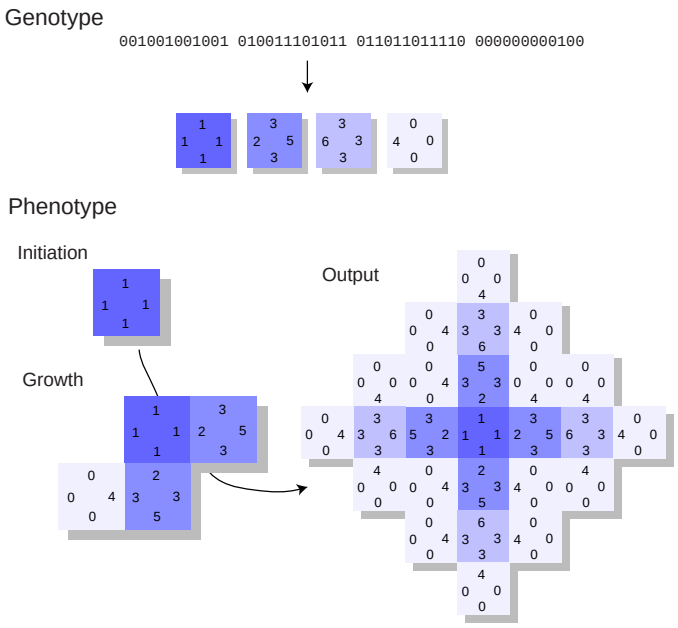

Figure 2.9: Illustration of the polyomino assembly model, adapted from [27]. A *genotype*, in the form of a ruleset of possible tiles, encodes for a polyomino *phenotype*, grown stochastically from an initial seed tile.

2.2.4 Patchy particle simulation

Besides discrete tile models, self-assembly can also be modelled using Molecular Dynamics simulations of rigid-body spheres called *patchy particles*. Each particle has a number of patches that bind when they come in contact with another complementary patch.

One such patchy particle simulator is included as part of the oxDNA package [29]. It was, for example, used by Romano et al., as seen in Figure 2.11, to verify the patchy interactions designed by their SAT-solver method [30].

In this thesis, the same patchy particle model is used to produce the results seen in Chapter 4. However, to account for the polycube requirement of patch orientation alignment, the model has been modified to include torsional interactions.

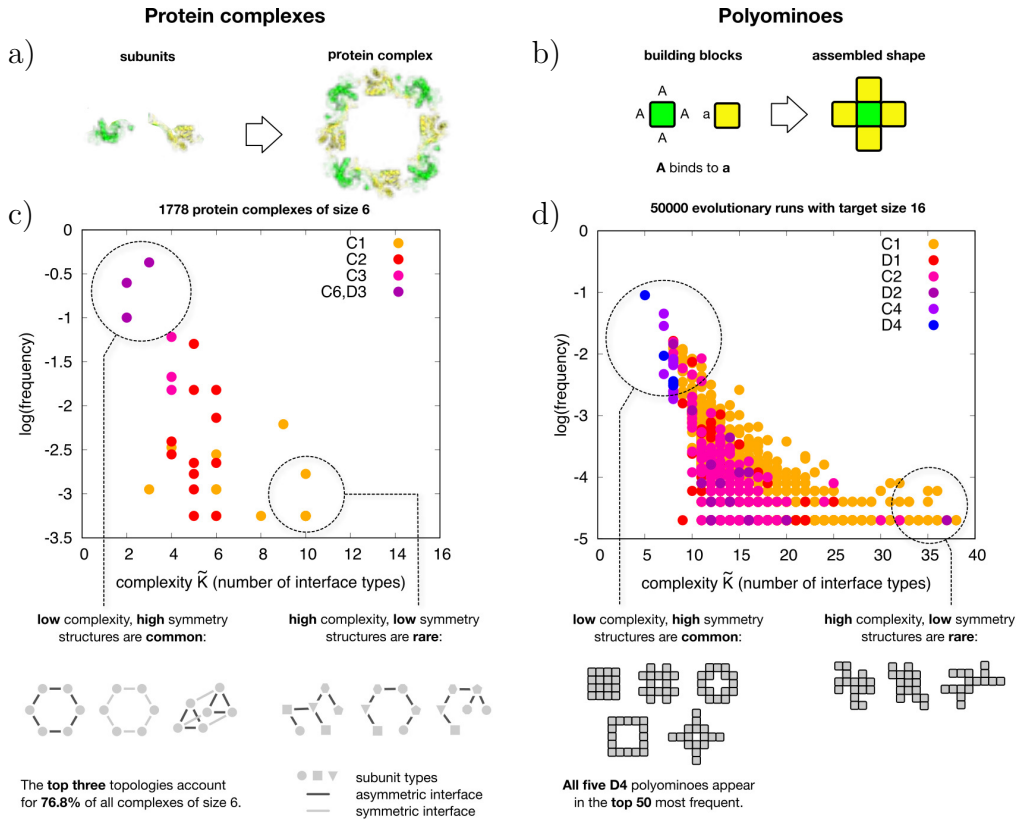

Figure 2.10: Frequent symmetry and simplicity through evolution, adapted from [28]. Both protein complexes (a) and polyominoes (b) self-assemble from individual units. **c)** Frequency of 6-mer protein complex topologies in the protein data bank, versus their complexity (measured as the number of interface types) **d)** Frequency versus complexity of polyominoes found in evolutionary runs with a fitness function seeking 16-mers.

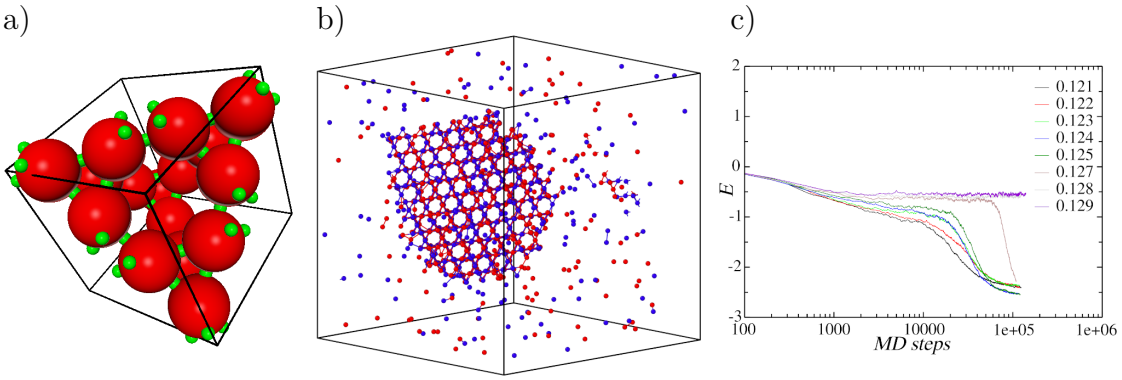

Figure 2.11: Patchy particle simulation, adapted from [30]. **a)** The unit cell of a tetrastack lattice build with patchy particles. **b)** Simulation snapshot of a forming tetrastack lattice. Note the free-flowing particles that have not yet attached the growing latttice they surround. **c)** Tetrastack particle energy plotted over simulation time for different temperatures. Sudden drops in energy correspond to nucleation events (where the lattices start forming).

3

Modular self-assembly of polycubes

Contents

As introduced in the previous chapter, there is an increasing interest within the field of DNA nanotechnology to create finite-sized multi-component objects. While some coarse-grained tile models exist, there remains a need for methods to quickly explore the assembly of multi-component 3D structures. This chapter describes my *polycube* model and details how it can be used to sample a large number of assembly rules, showing that some polycube shapes are significantly more common than others. The model supports 3D polycubes and 2D polyominoes, both explained in the following sections. The following chapter (Chapter 4) will show how to obtain

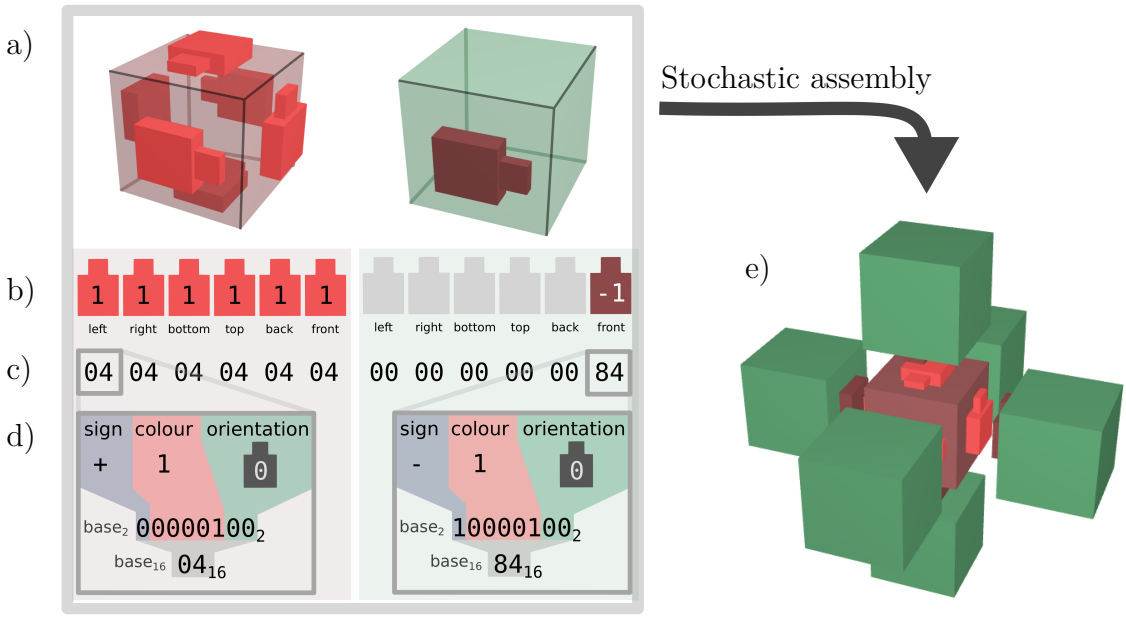

Figure 3.1: Illustration of the polycube model and notation, examplified with the rule 040404040404000000000084. Compare this to the polyomino model in Figure 2.9. **a)** 3D representation of the species in the rule. **b)** Rule depicted as a list of the patches in each species. The empty patches (colour 0) in the green species are just shown with their orientations. All orientations are 0 in this rule, since changing them would not change the output. **c)** Hexadecimal representation of the rule, shown decoded in **d)**, where every 2-digit hexadecimal number represents a patch. Converted to a 8-bit binary number, first bit encodes the sign, the next five bits the colour ($[0, 31]$), and the final two bits encode the orientation ($[0, 3]$). **e)** Fully assembled polycube output. The assembly used one copy of the first species (red) and six copies of the second (green). The assembly finished since no further cubes could be added.

the simplest assembly rule for any given polycube shape.

3.1 The polycube model

A *polycube* consists of multiple equally-sized cubes connected by their neighbouring faces (a three-dimensional analogue to how polyominoes are squares connected by their neighbouring edges). In the model presented here, a polycube is stochastically self-assembled according to a specified *rule*, defining a set of available cube species. Each species describes a type of cube that can be present in the polycube, so cubes belonging to the same species are always identical. See, for example, Figure 3.1, where an input rule with two species assembles into a double-cross output polycube

with seven cubes.

Each species has six patches; one on each face of the cube, and each patch has a "colour" and an orientation. The colour is indicated by a signed integer and the orientation is one of four possible rotations:  (0),  ($\frac{\pi}{2}$),  (π) or  ($\frac{3\pi}{2}$), saved as an integer 0, 1, 2, or 3.

For each of the patches, facing left $[-1, 0, 0]$, right $[1, 0, 0]$, bottom $[0, -1, 0]$, top $[0, 1, 0]$, back $[0, 0, -1]$ and front $[0, 0, 1]$ respectively, the default (0) orientation () corresponds to the vectors $[0, -1, 0]$, $[0, 1, 0]$, $[0, 0, -1]$, $[0, 0, 1]$, $[-1, 0, 0]$, and $[1, 0, 0]$.

A patch can bind to another patch if and only if they have the opposite colour and the same (global) orientation. In Figure 3.1, the patch color 1 is shown as bright red while the complementary -1 is a darker red colour. If the patch colour is zero, the patch is shown as empty and will not bind to anything.

The model can be expanded to more complicated colour interaction matrices or changed to pair odd integers with each subsequent integer as in the polyomino model [26, 27] described in Section 2.2.3.

By constraining the input space not to use the back and front patches, while orienting the remaining patches to point towards the front, the output space will instead be 2D polyominoes.

3.1.1 String representation

Polycube rules can be described in a hexadecimal representation, as seen in the Figure 3.1.c). Each species is described by 12 hexadecimal digits, two digits per patch. The two hexadecimal digits are then converted into eight binary digits (bits). The first six bits represent the patch colour as a signed integer (allowing for decimal values 0-31); the remaining two encodes one of the four possible patch orientations.

Alternatively, a less compact but more human-readable decimal notation can be used, with each patch written as `c:o` (where `c` is the integer colour and `o` is the integer orientation) and delimited by vertical bars (`|`). Finally, each species is delimited

by underscores (_). For example, the rule used in Figure 3.1, "040404040404 000000000084", would be written as "1:0|1:0|1:0|1:0|1:0|1:0_|||||-1:0"

3.1.2 Stochastic self-assembly

The stochastic self-assembly of a polycube starts by placing a cube from one of the available species as a seed at the origin. If the assembly mode is *seeded*, it will always use the first species in of rule. If the assembly mode is *unseeded*, the seed species is instead chosen at random. Once a cube has been added to the assembly, all available neighbouring positions are added to a list of possible moves.

Moves are then processed until the list of moves is empty, or the polycube grows beyond a specified size, at which point it is considered *unbounded*. Figure 3.2.a) shows an example of an unbounded structure tiling the plane using two species.

While the list of moves is not empty, a random move is chosen at each step. The input rule is then randomly searched for a species fitting the move. Cubes can be rotated to fit, and if a fitting species is found, the corresponding cube is added to the assembly. If there is no fit to be found, the move is discarded.

The assembly is repeated n_{times} times (default 100), and the outputs are compared for equality (allowing rotation) in order to determine if the rule is *deterministic*. Figure 3.2.b) shows a non-deterministic structure, where the blue "neck" species can bind to itself. Thus, the output depends on how many cubes from the blue species bind before a green species cube stops the growth. Both species are as likely to bind, so the probability of assembling a neck with l blue cubes is 2^{-l}.

Unbounded or non-deterministic structures are considered *undefined* and are not included in the sampling result. This is in line with the earlier polyomino model and comparable to how unbounded protein complexes or non-deterministically folding proteins are highly deleterious [28].

It could be argued, instead of first picking a random move and then randomly trying all available species to find a fit, that one should pick both a move and a species at random until a fit is found. While this would take a longer time, it would avoid biasing the assembly toward unlikely assembly results, where a move is

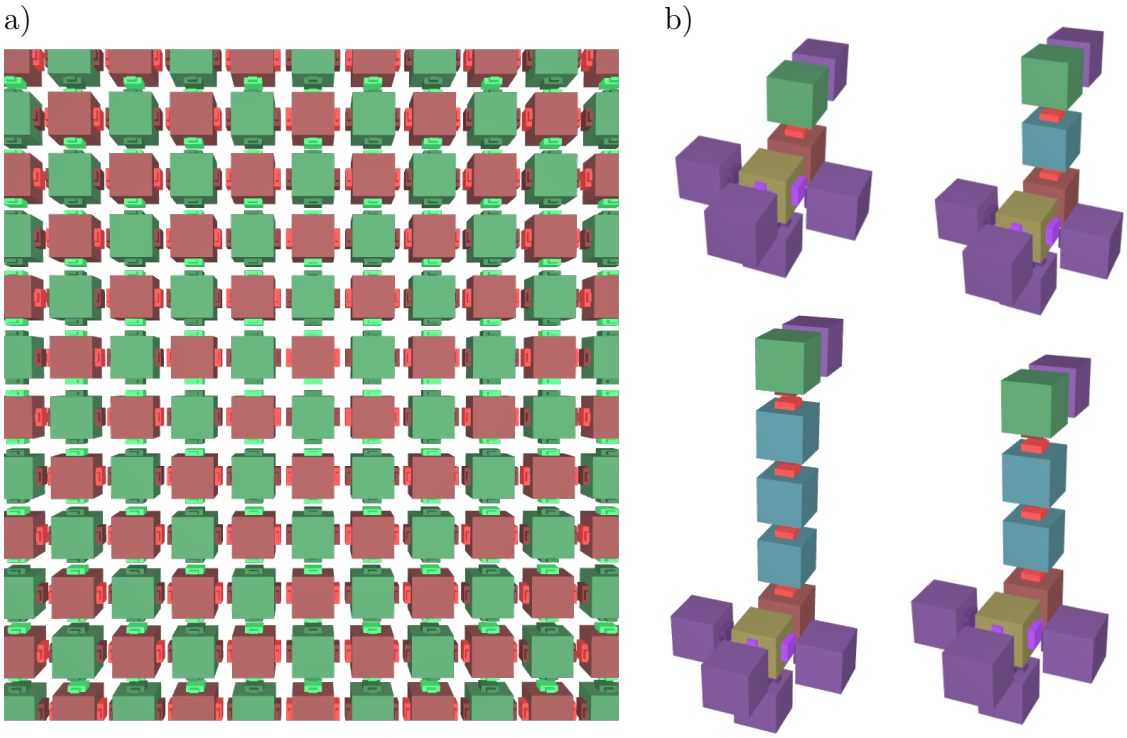

Figure 3.2: Examples of undefined assemblies. **a)** Unbounded assembly that tiles the plane using two species (05050a08000085858a880000), **b)** An undeterminisic assembly of a "giraffe duck" with a neck that can have a different length each time it is assembled (00000006008b00008600000c000000028c00080c0c000c0c048600000000).

picked that would otherwise usually be blocked by more likely surrounding cubes. However, since only deterministic and bounded rules are of interest, this would only affect the end result in the cases where the bias is strong enough and n_{times} is low enough to incorrectly make the rule seem deterministic.

For an illustration of the model, let us return to Figure 3.1. The example in the figure is a three-dimensional "double-cross" structure created from a rule of size 2. The initial seeding cube belongs to the first species, enabling six additional cubes, all belonging to the second species, to bind at each patch. The patches bind since their colours, 1 and-1, are opposites. After all six outer cubes have bound, there are no remaining possible moves, and thus the polycube stops growing. Since the growth stops, this particular polycube is bounded at a size of seven cubes. Furthermore, since the rule gives the same polycube every time it is evaluated, the polycube is deterministic.

3.1.3 Implementation

The polycube assembly model is implemented in two versions: one browser-based implementation in JavaScript (`https://akodiat.github.io/polycubes`) for outreach activities and accessible visualisation and one C++ implementation for fast rule evaluation. The C++ code also includes a Python binding for simplified analysis. More details on the code can be found in Appendix A.

3.2 Sampling the space of assembly rules

Trying all inputs in a brute-force approach is unfeasible for most input spaces. For a 3D polycube space with $\widetilde{K}_s$ species and $\widetilde{K}_c$ colours, four possible patch orientations, and six patches per species, we get:

$$\left|I^{3d}_{\widetilde{K}_s,\widetilde{K}_c}\right| = (4 \times (1 + 2\widetilde{K}_c))^{6\widetilde{K}_s}$$

Even for a relatively small values of $\widetilde{K}_s = 3$ species and $\widetilde{K}_c = 2$ colours, we get $I^3_{2,3} \approx 2.62 \times 10^{23}$.

For 2D, it is a bit easier since there are only four patches per species and a fixed patch orientation:

$$\left|I^{2d}_{\widetilde{K}_s,\widetilde{K}_c}\right| = (1 + 2\widetilde{K}_c)^{4\widetilde{K}_s}$$

However, even if the space of possible input rules is too large to explore fully, we can still get an idea of how likely it is for an input rule to map to a particular output shape through uniform sampling.

This was done by sampling and assembling a large number of random rules. As described in Section 3.1.2, rules growing larger than 100 cubes were discarded as unbounded, while those remaining bounded were re-assembled 100 times to ensure they assembled deterministically.

Deterministic and bounded output was then grouped by their shapes, counting the number of times each given shape occurs. For each rule found to produce a given shape, three different complexity measures were calculated:

$\widetilde{K}_s$ - The number of species used.

$\widetilde{K}_c$ - The number of colours used.

$\widetilde{K}_{lz}$ - The length of the binary rule after Lempel–Ziv compression [31].

To more fairly compare them, each rule was simplified before the complexity measure was calculated. This simplification was done by removing all patches without a complementary colour in the same rule, as well as setting the orientation of all empty (zero-coloured) patches to zero. Finally, the colour indices were updated to avoid any gaps in the numbering.

3.2.1 Polyomino reference sampling

In order to verify the model against earlier polyomino results [28], a large reference sampling of the 2D space was performed. Limiting the input space to 16 species and 31 colours ($I^{2d}_{16,31}$), one billion (10^9) random rules were assembled using the seeded assembly mode.

Figure 3.3: Proportion of valid rules when sampling $I^{2d}_{16,31}$. From a total of 1,000,000,000 sampled rules, 44,545,570 were found to be valid, while 871,155,425 were unbounded and 84,299,005 were non-deterministic

Let us start by looking at the sampled input space. How much of the input maps to valid output? As can be seen in Figure 3.3, a large number of the sampled rules were either unbounded or not assembling deterministically. The boundary between unbounded and non-deterministic assemblies is not clear, since some input also can be both (but is classified as either one or the other).

Figure 3.3 shows that most of the valid rules assembled into smaller shapes, with over 60 per cent (27 million) being 1-mers (note that the y-axis is logarithmic).

With the input space investigated, we can also check how much of the output space we managed to sample. In Figure 3.5, the blue bars show how many

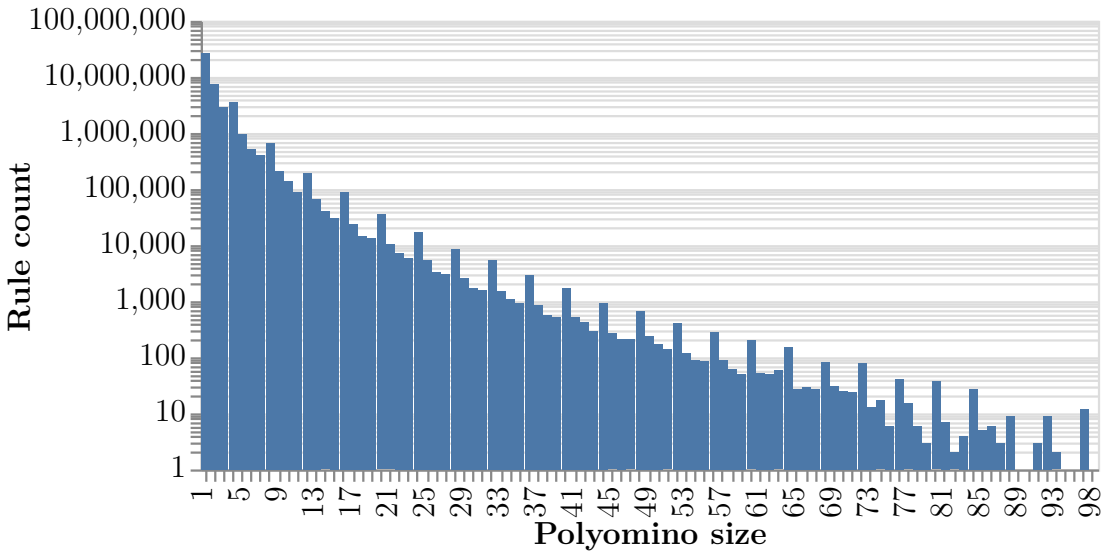

Figure 3.4: Number of input rules per output size. Distribution of polyomino sizes of 10^9 sampled rules in $I^{2d}_{16,31}$.

polyominoes were found of each size, while the red line shows the total number of polyominoes that exist of that size (obtained from the On-line Encyclopedia of Integer Sequences [32, 33]). The sampled input space $I^{2d}_{16s,31c}$ should allow for all 16-mers to be assembled, since there are enough species and colours for fully addressable assemblies (assigning each cube its own species and each connection its own colour). Thus, the blue bars would follow the red line until size 16 if the complete input space was enumerated. Figure 3.5 shows that we find at least the correct order of magnitude up until about 10-mers, after which polyominoes of larger sizes are found in decreasing numbers. Note how some polyomino sizes have a much higher count than their neighbours, indicating a bias for shapes of that size.

3.2.2 Main samplings

The main sampling was done to compare results between 2D and 3D, as well as the seeded versus unseeded assembly modes. All these samplings were done with 10^8 samples each in the $I_{5s,31c}$ space.

The reason for only using five species per rule is to ensure enough valid output for stochastic 3D, which, as can be seen in Figure 3.6 is relatively low.

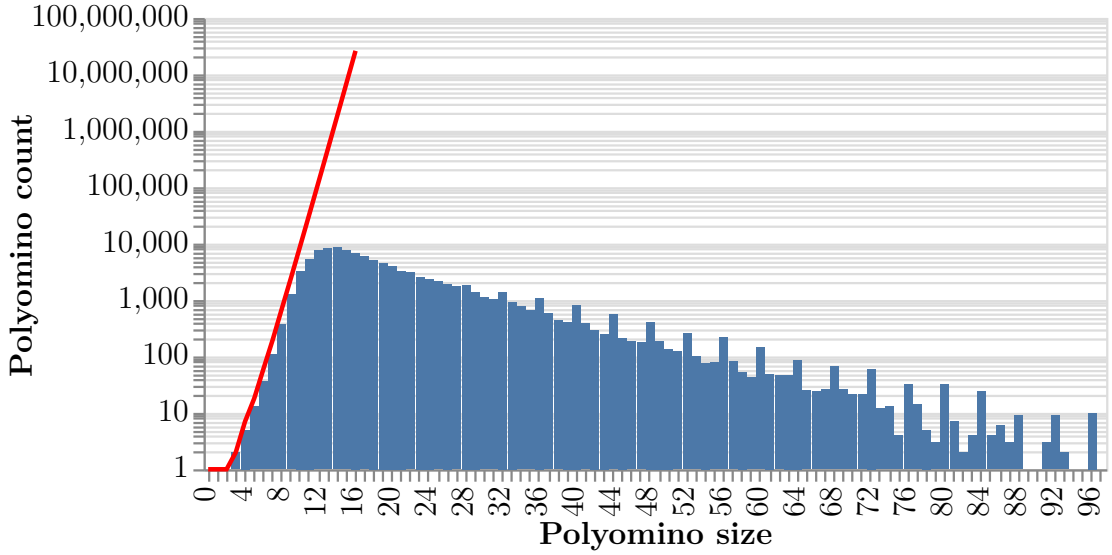

Figure 3.5: Number of output polyominoes per output size. Count of unique output polyominoes found sampling 10^9 input rules in $I^{2d}_{16s,31c}$. The red line shows the actual number of polyominoes of each size, obtained from OEIS A000988 [32, 33].

In general, the seeded assembly mode is expected to generate more valid rules than when the seed is random, as Figure 3.6 shows . An input rule that depends on the seed can assemble different shapes every time and will be deemed non-deterministic, while the same rule would be considered valid for seeded assembly. With the third dimension, there is also more orientations and more patches per cube to attach to, leading to the significantly fewer valid 3D rules seen in the figure.

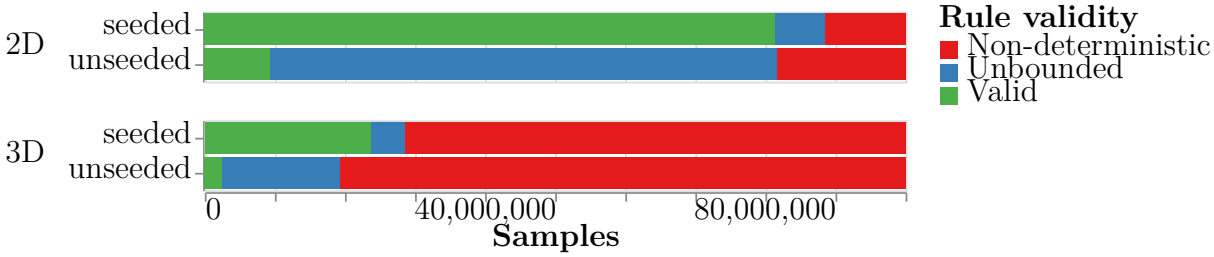

Figure 3.6: Proportion of valid rules when sampling $I_{5s,31c}$ for seeded and unseeded assembly in both 2D and 3D.

Figure 3.7 shows the distribution of output sizes for the main samplings. Once again, we can see that the number of shapes found initially follow the same order of magnitude as the total number of existing shapes for each size. Here we can also see even more clearly how shapes of certain sizes show up much more than

their neighbours. For example, 1528 36-mers were found in the seeded 3D sampling, while only 30 35-mers and 58 37-mers were found. Such spikes are found for 24-mers, 28-mers, 32-mers, 36-mers, 40-mers, 44-mers, and 48-mers, possibly showing common symmetrical structures growing by four cubes at a time.

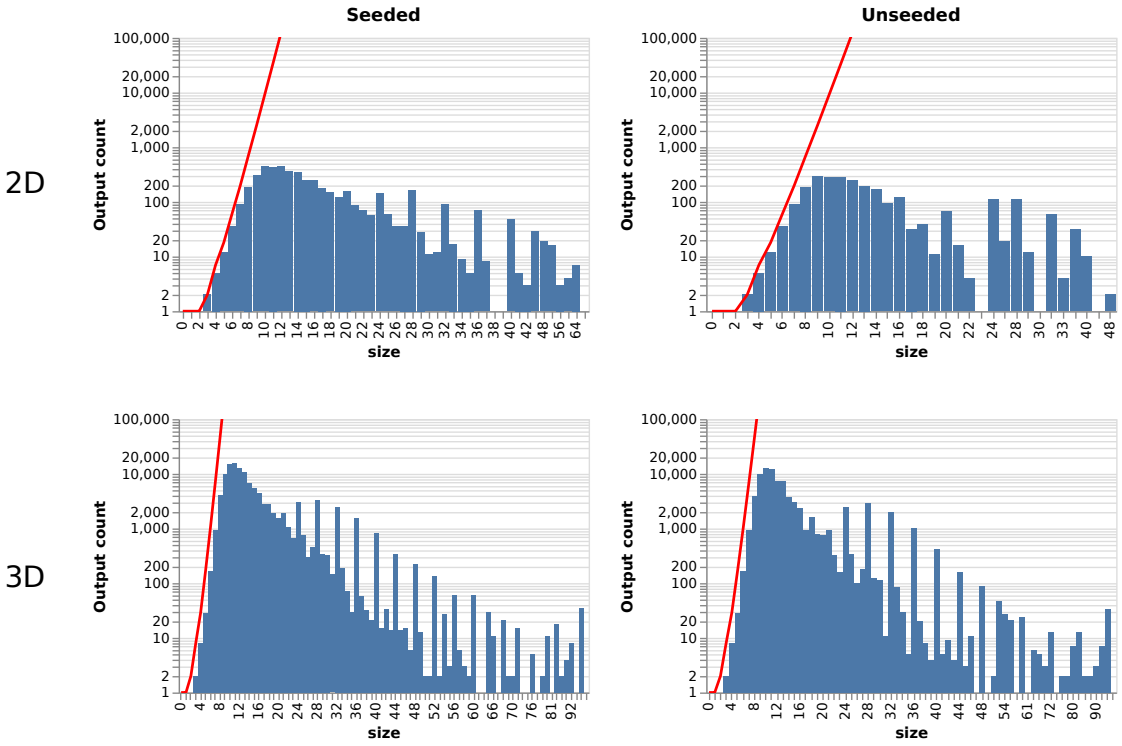

Figure 3.7: Distribution of output sizes when sampling $I_{5s,31c}$ for seeded and stochastic in both 2D and 3D. The red line shows the actual number of polyominoes of each size, obtained from OEIS A000988 and A000162 [32, 33] respectively.

3.3 Symmetry

For polycube and polyomino symmetry, we care about rotation and reflection symmetry on the lattice. A 2D polyomino has four possible rotations, corresponding to the matrices $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$, $\left(\begin{smallmatrix}0&-1&0\\1&0&0\\0&0&1\end{smallmatrix}\right)$, $\left(\begin{smallmatrix}-1&0&0\\0&-1&0\\0&0&1\end{smallmatrix}\right)$, and $\left(\begin{smallmatrix}0&1&0\\-1&0&0\\0&0&1\end{smallmatrix}\right)$. The first one is simply the identity matrix and can be ignored. However, if the lattice coordinates remain unchanged after multiplication with any of the others, it means that the polyomino has rotational symmetry. A polyomino also has four possible reflections: one across the x-axis, one across the y-axis, and two along the diagonals. Similarly, a 3D

polycube has 24 possible orientations (including identity), as well as reflections around the x-y, x-z, y-z and diagonal planes.

In the simplified context of these lattice shapes, we assign 2D symmetry groups as follows:

```
if reflsymms == 2:
  group = 'D4'
elif rotsymms == 3:
  group = 'C4'
elif reflsymms == 1:
  group = 'D2'
elif rotsymms == 1:
  group = 'C2'
elif rotsymms == 0:
  group = 'C1'
elif reflsymms == 0:
  group = 'D1'
```

3.4 Results

With the sampling details and the analysis methods explained, we will now move on to the main results. Let us start by looking at the different shapes found. Figure 3.8 shows a gallery of the 16-mer polyominoes most commonly found when sampling $I^{2d}_{16,31}$. Each shape is scaled proportional to the frequency at which it was found, showing that, even within a given size, some shapes are much more common than others.

Similarly, Figure 3.9 shows a gallery of the 8-mer polycubes found when sampling $I^{3d}_{5,31}$. Here, the difference in frequency between the shapes was so significant that they instead had to be scaled proportional to the *natural logarithm* of their frequency in order to be visible in the same figure.

Note how the most common shapes seem to be relatively simple, symmetrical or modular, concepts that will be explored further in coming sections.

3.4.1 Frequency and rank

If we plot the frequency of the shapes against their relative rank (with rank 1 being the most frequent, 2 being the second-most frequent, et cetera.), the scaling

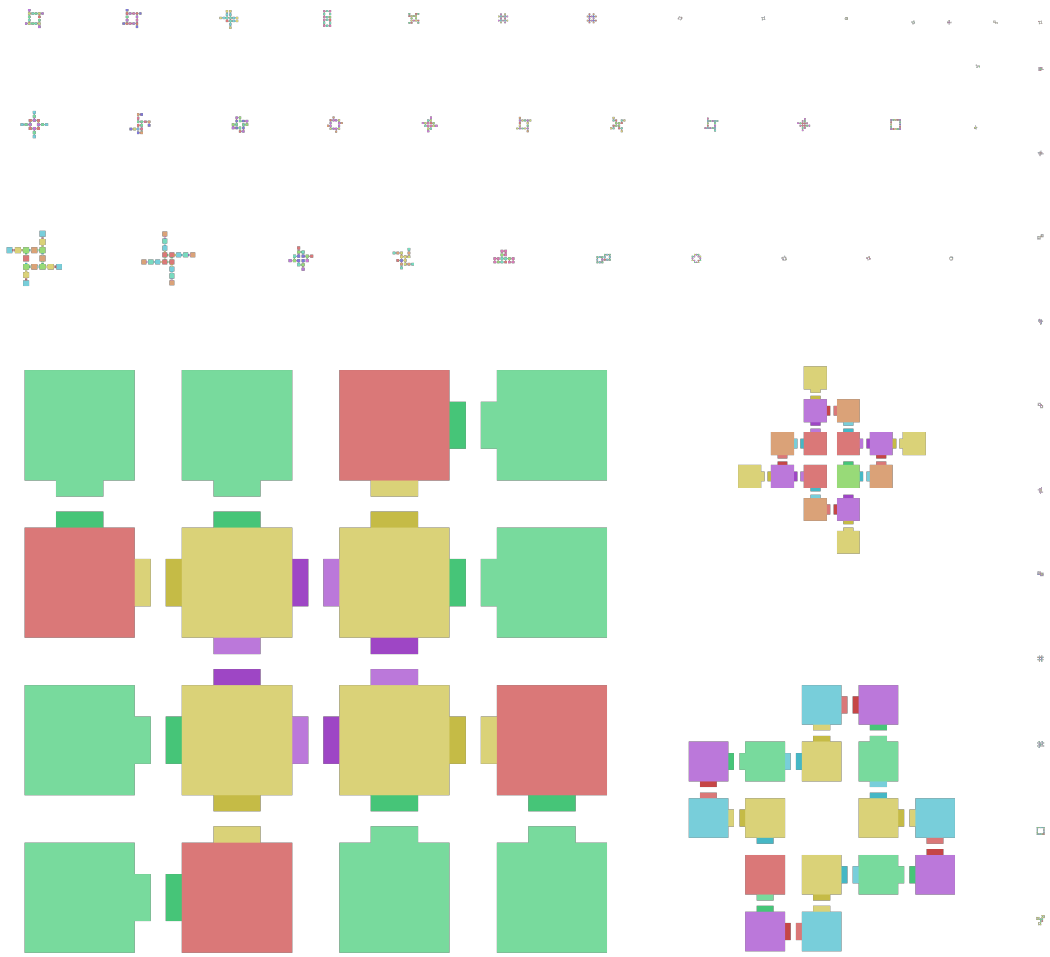

Figure 3.8: The 50 most common 2-dimensional 16-mers, scaled proportional to their frequency. Found while sampling the $I^{2d}_{16,31}$ input space.

is quite significant, as seen in Figure 3.10.

The sharp decrease for the lowest-ranking shapers is a sampling artefact. With a sample size of 10^8, a frequency of less than 10^{-7} means that the shape was found fewer than 10 times. Some rare shapes can be found a few times through random chance, giving them a much higher frequency than they would have during a larger sampling.

The sharp decrease among the highest-ranking shapes, however, once again shows how some shapes are much more common than others. Let us see if we can find any features of these shapes that stand out.

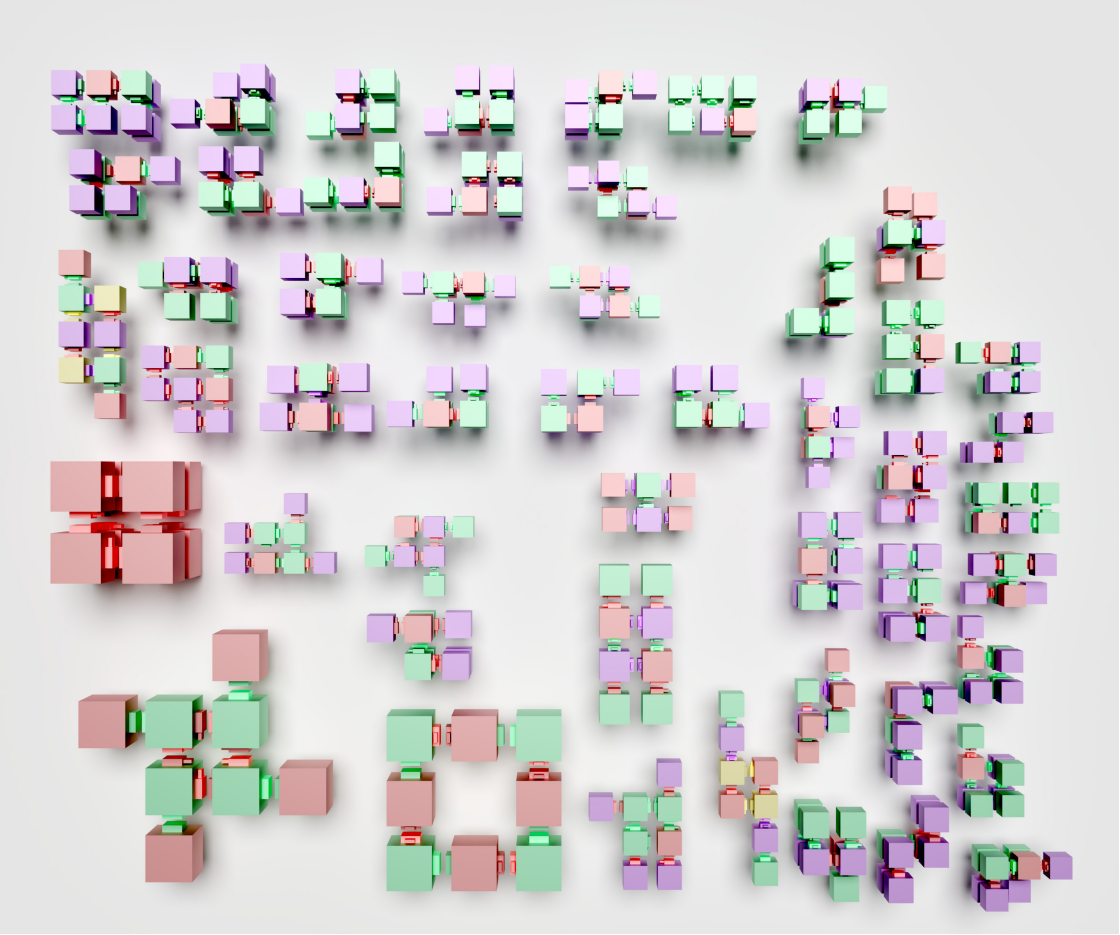

Figure 3.9: The 50 most common three-dimensional 8-mers, scaled proportional to the natural logarithm of their frequency.

3.4.2 Frequency and complexity

Let us return to the complexity bias presented in Section 2.2.3. Can we show such a trend for the current model as well? Keeping the parameters similar, we start by examining the results of the reference sampling described in Section 3.2.1. As can be seen in Figure 3.11, there is a log-linear relationship between the probability of finding a rule that assembles into a particular structure and the information needed to specify the structure, as predicted in [11, 34].

Comparing Figure 3.11 with Figure 2.10, it is clear that the same simplicity bias is present, reproducing the result from [28] with the polycube model implementation presented in this thesis. Note how high-frequency low-complexity shapes tend to be more symmetric. The unusually low frequency of the two C2 polyominoes

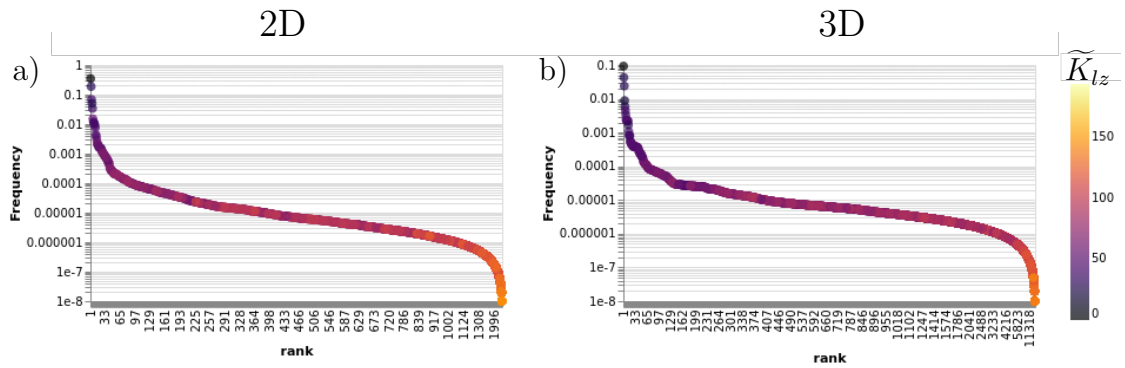

Figure 3.10: Frequency vs rank for 10^8 samples of $I_{8s,31c}$ in both 2D (a) and 3D (b). Note that the frequency axis is logaritmic. Each point is a unique shape, coloured by its complexity ($\widetilde{K}_{lz}$).

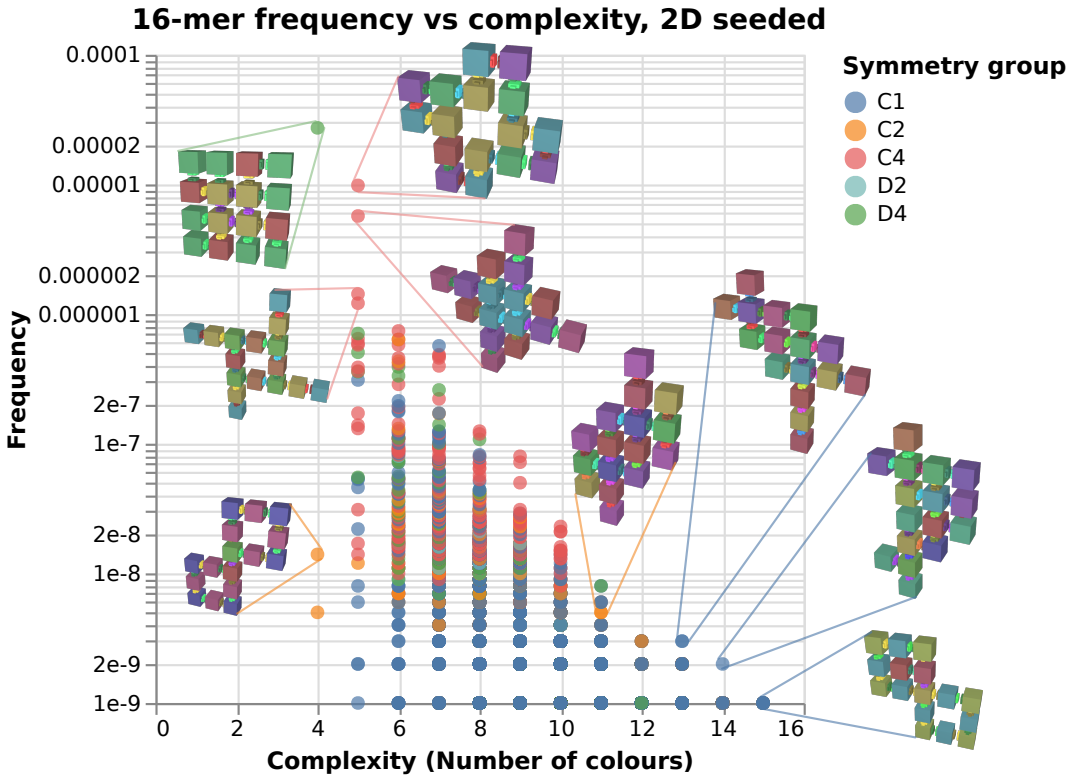

Figure 3.11: Frequency vs complexity ($\widetilde{K}_c$) of 16-mers found when sampling $I^{2d}_{16,31}$. Each point represents a unique polyomino shapes, some of which are visualised.

with $\widetilde{K}_c = 4$ is an example of how the $\widetilde{K}_c$ measure is an imperfect proxy for Komologrov complexity; both have higher complexity using alternative measures ($\widetilde{K}_s$=11 and $\widetilde{K}_{lz} \approx 211$).

Extending our result into three dimensions, as well as using both seeded and unseeded assembly, let us now look at the results from the main sampling, introduced

in Section 3.2.2.

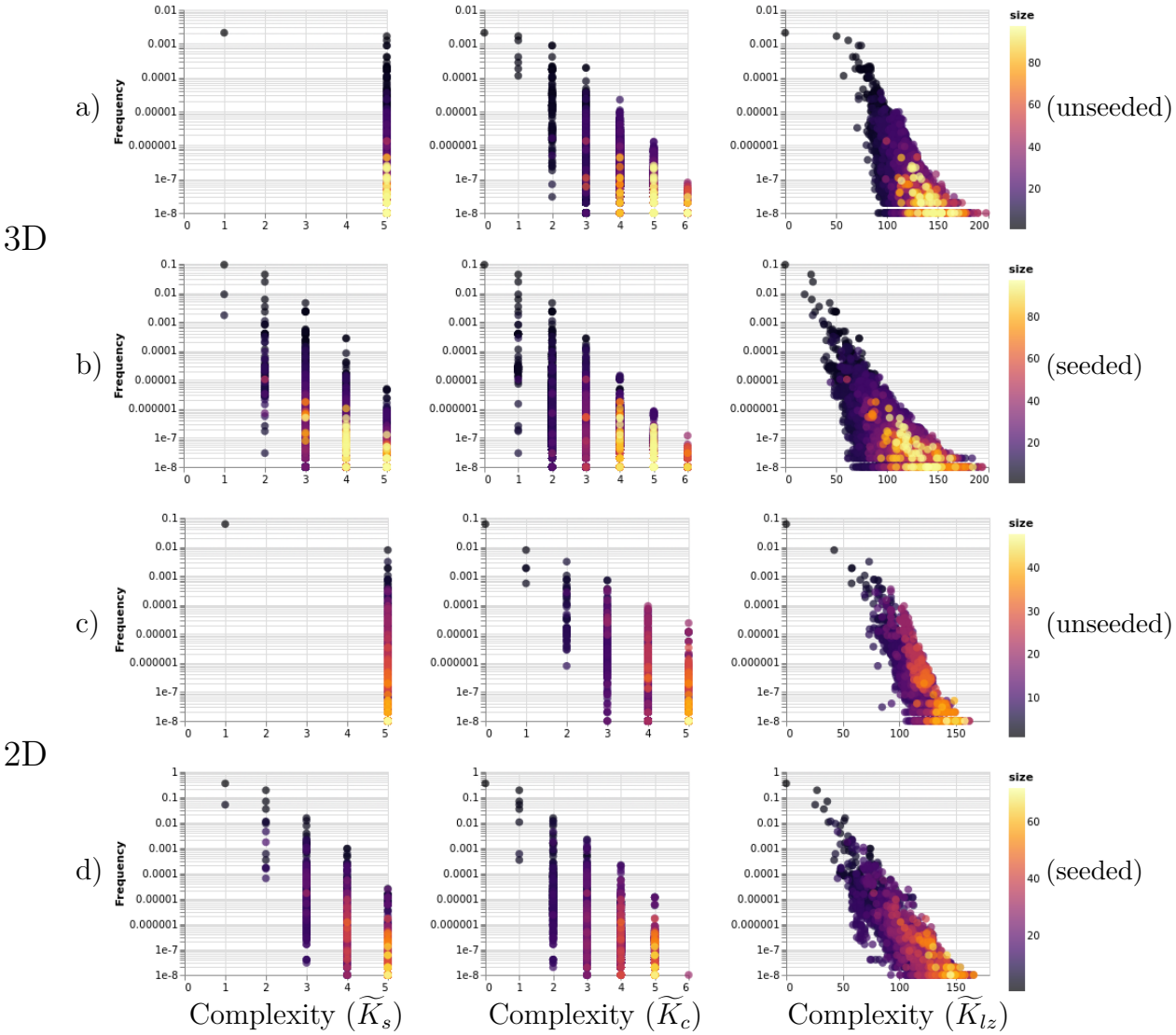

Figure 3.12: Frequency of shapes found when sampling $I_{5s,31c}$, versus different measures of their complexity. Each point is a unique polycube (or polyomino) shape, coloured proportional to its size. Each column shows a different complexity measure, as proxies for Komologrov complexity. The left column measures the minimum number of species required to assemble each shape, the middle instead shows the minimum number of colours required, and the right column is the shortest Lempel–Ziv-compressed rule. **a)** Unseeded assembly in 3D. **b)** Seeded assembly in 3D. **c)** Unseeded assembly in 2D. **d)** Seeded assembly in 2D. Note that the vertical axis (Frequency) is logarithmic.

As seen in Figure 3.12, the simplicity bias is still present for 3D polycubes, even using alternative measures of the complexity. The measures provide different resolution, but all still show the frequency of a shape bounded by its complexity.

One apparent break from the trend shown is the mostly constant $\widetilde{K}_s$ values for

unseeded samplings (Figure 3.12.a) and 3.12.b)); almost all the output shapes use the maximum number of species. This can be explained as a consequence of the determinism check. A seeded rule can afford to have a few non-binding species that are simplified away and not counted, as long as none of them are the seed. But with unseeded assembly, any species can be the seed; so such a rule would be deemed non-deterministic and not included. Thus, for the unseeded results, we have to rely on the other two complexity measures.

3.4.3 Modularity

Besides symmetry and complexity, we can also look at how the modularity of a shape can limit its frequency. We use a simple *modularity index*, defined as the polycube size divided by the number of needed species ($\widetilde{K}_s$). Basically, a modular shape is one where modules are reused many times. For the reasons detailed above, it only makes sense to look at the modularity of seeded assemblies, since $\widetilde{K}_s$ is otherwise mostly constant.

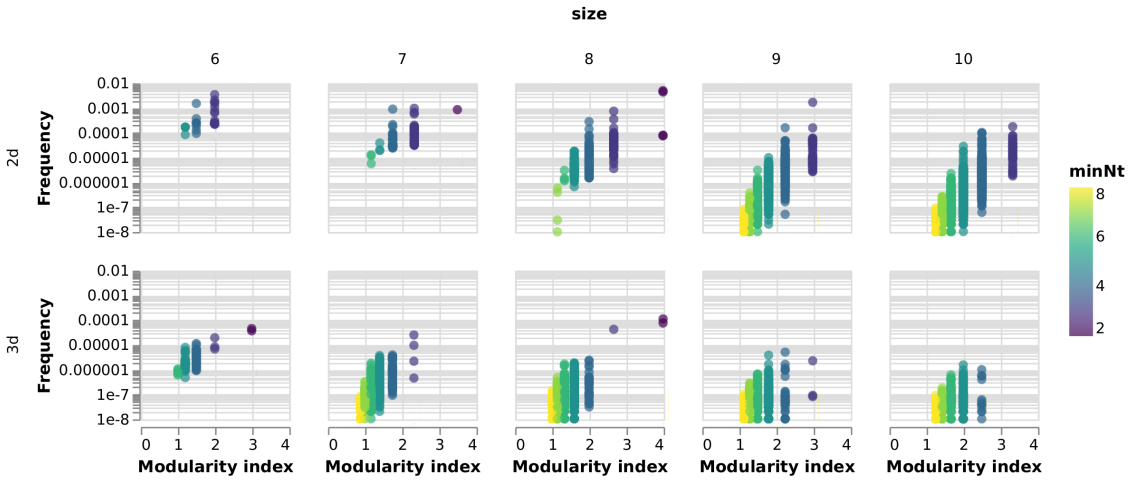

Figure 3.13: Frequency vs modularity for a selection of sizes. The top row shows 2D polyominoes, while the bottom row shows the 3D polycubes. From 1e8 samples of $I_{8s,31c}$.

Figure 3.13 shows how the modularity correlates with the frequency for a selection of sizes in both two and three dimensions. For a fixed size, the modularity index is really only a scaled inverse of the $\widetilde{K}_s$ complexity, but it is still helpful to see how modular shapes have a significantly higher frequency.

We have now shown how simple designs can be found through random sampling, but how would one go about designing a rule that assembles a specific shape? And does it matter if the rule is simple or not? That will be covered in the coming chapter.

4

Designing polycube assembly rules

Contents

In the previous chapter, we saw how to map an input rule into an output polycube shape. However, the reverse problem is just as significant; given a target shape, how do you find a rule that assembles it?

A trivial solution would be to use *fully addressable assembly*: simply assign a unique species to each cube and a unique colour to each pair of adjacent patches. This is similar to what was done for DNA bricks (Section 2.1.1), where every brick tile is unique. However, as was seen in Chapter 3, many shapes have alternative solutions requiring significantly fewer unique components. See Figure 4.1, where a

square tetromino is shown to have a variety of inputs assembling it, from the minimal solution with just a single species and one colour, up to the fully addressable solution with four species and four colours ($\widetilde{K}_s = \widetilde{K}_c = 4$). The intermediate solutions are not necessarily deterministic in terms of which position gets which species, but they will always assemble into the same shape.

Meanwhile, the empty red region in Figure 4.1.b) shows combinations of $\widetilde{K}_s$ and $\widetilde{K}_c$ for which a solution is not possible. For example, if you use a single species, you cannot use more than one colour.

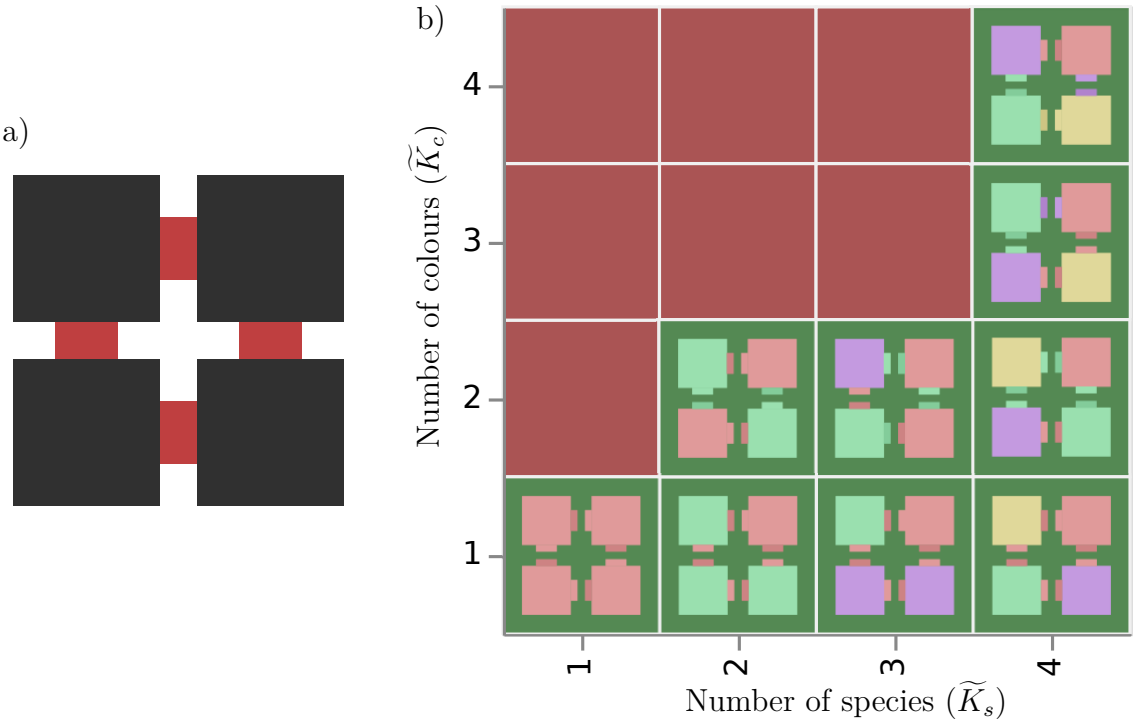

Figure 4.1: 2×2 square polyomino assembled with different levels of complexity. **a)** Scematic of the input shape, consisting of four connected tiles. **b)** The green region shows possible assembly solutions, from the *minimal solution* using a single species and a single colour (bottom left), to the *fully addressable solution* using four species and colours (top right). The red region lacks solutions.

So, how do we find these alternative and simpler input rules for a shape? Surely, there must exist a better method than sampling the space of all rules (as done in Chapter 3)? This chapter presents an approach where *satisfiability solving* is used to determine if a shape can be assembled from a given number of colours and species, thus automatically filling in solution landscapes such as the one shown

in Figure 4.1. Furthermore, a complementary approach of substituting similar species is also described, in Section 4.3.

4.1 Satisfiability solving

Building upon a method for determining patchy particle interactions for unbounded structures [30], it is possible to formulate and solve satisfiability problems for the bounded polycube structures.

In essence, we formulate a boolean expression that, if true, means it is possible to assemble a given polycube topology using a given number of colours and species. We can then use a satisfiability solver to check if that expression is indeed solvable and, if it is, extract an assembly rule from the solution.

4.1.1 Boolean expressions

The boolean expression is written in conjunctive normal form (CNF), where variables are composed into clauses using *NOT* ($\neg$) and *OR* ($\vee$) operators and where the clauses are joined by *AND* ($\wedge$) operators. As a simple example, see the expression below:

$$(\neg x_{rain} \vee x_{umbrella} \vee \neg x_{walk}) \wedge (\neg x_{rainbow} \vee x_{rain}) \wedge (\neg x_{rainbow} \vee x_{sunny})$$

The first clause is $true$ for all values except when $x_{rain} = true$, $x_{umbrella} = false$ and $x_{walk} = true$; so the solution of taking a walk in the rain without an umbrella is forbidden (allthough I would personally prefer a good coat). This could also be written as an *implication*: $x_{rain} \wedge x_{walk} \implies x_{umbrella}$.

The following two clauses in the example above are the CNF form of another implication: $x_{rainbow} \implies x_{rain} \wedge x_{sunshine}$, stating that a rainbow implies that we have both rain and sunshine (we cannot have a rainbow without rain or without sunshine). The full expression is satisfiable, for example, if we set $x_{sun} = true$, $x_{rain} = false$, $x_{walk} = false$, $x_{umbrella} = false$, and $x_{rainbow} = false$; ignoring the sunshine walk and remaining inside to work.

4.1.2 Polycube formulation

For the polycube problem, we introduce the following variables:

$x^A_{l,p,o}$ (patch p at position l has orientation o)

$x^B_{c_i,c_j}$ (colour c_i is compatible with colour c_j)

$x^C_{s,p,c}$ (patch p on species s has colour c)

$x^D_{p_1,o_1,p_2,o_2}$ (patch p_1 with orientation o_1 binds to patch p_2 with orientation o_2)

$x^F_{l,p,c}$ (patch p at position l has colour c)

$x^O_{s,p,o}$ (patch p on species s has orientation o)

$x^P_{l,s,r}$ (position l is occupied by species s rotated by r)]

We then formulate clauses to constrain the problem, seen in Table 4.1. Clauses (i)-(vii) are the same as in [30] while the remaining are added, together with variables x^D, x^A and x^O abpve, to include *torsional restrictions*, meaning that patches need to bind at a compatible orientation (compared to being allowed to rotate freely).

4.1.3 On the importance of torsional interactions

It would certainly be possible to use the solver without any constraints on the patch orientations (as it was done in [30]). However, if we wanted to use the stochastic assembler from Chapter 3, orientations would have to be assigned randomly, resulting in a combinatoric explosion of additional assembly paths.

More importantly, the assembly should benefit from torsional patch interaction (for 2D polyominoes, this corresponds to the requirement that tiles can be rotated in the plane but not flipped). Figure 4.2 shows two versions of a simple rule, the only difference being the orientation of a single patch. While co-operative binding might, in such a case, benefit the desired $\frac{\pi}{2}$ square assembly, the self-limiting ability would be significantly improved if the patches were torsionally rigid.

	Clause	Boolean expression
(i)	$C^{B}_{c_i,c_j,c_k}$	$\neg x^{B}_{c_i,c_j} \vee \neg x^{B}_{c_i,c_k}$
(ii)	C^{C}_{s,p,c_k,c_l}	$\neg x^{C}_{s,p,c_k} \vee \neg x^{C}_{s,p,c_l}$
(iii)	$C^{P}_{l,s_i,r_i,s_j,r_j}$	$\neg x^{P}_{l,s_i,r_i} \vee \neg x^{P}_{l,s_j,r_j}$
(iv)	$C^{BF}_{l_i,p_i,c_i,l_j,p_j,c_j}$	$\left(x^{F}_{l_i,p_i,c_i} \wedge x^{F}_{l_j,p_j,c_j}\right) \Rightarrow x^{B}_{c_i,c_j}$
(v)	$C^{rotC}_{l,s,r,p,c}$	$x^{P}_{l,s,r} \Rightarrow \left(x^{F}_{l,p,c} \Leftrightarrow x^{C}_{s,\phi_r(p),c}\right)$
(vi)	C^{alls}_{s}	$\bigvee_{\forall l,r} x^{P}_{l,s,r}$
(vii)	C^{allc}_{c}	$\bigvee_{\forall s,p} x^{C}_{s,p,c}$
(iix)	C^{O}_{s,p,o_k,o_l}	$\neg x^{O}_{s,p,o_k} \vee \neg x^{O}_{s,p,o_l}$
(ix)	$C^{DA}_{l_i,p_i,c_i,l_j,p_j,c_j}$	$\left(x^{A}_{l_i,p_i,c_i} \wedge x^{A}_{l_j,p_j,c_j}\right) \Rightarrow x^{D}_{p_i,c_i,p_j,c_j}$
(x)	$C^{rotO}_{l,s,r,p,o}$	$x^{P}_{l,s,r} \Rightarrow \left(x^{A}_{l,p,o} \Leftrightarrow x^{O}_{s,\phi_r(p),o}\right)$

Table 4.1: SAT clauses. (i) Each colour is compatible with *exactly one* colour. (ii) Each patch has *exactly one* colour. (iii) Each lattice position contains a single species with an assigned rotation. (iv) Adjacent patches in the lattice must have compatible colours. (v) Patches at a lattice position are coloured according to the (rotated) occupying species. (vi) All $\widetilde{K_s}$ species are required in the solution. (vii) All $\widetilde{K_c}$ patch colours are required in the solution. (iix) Each patch is assigned *exactly one* orientation. (ix) Adjacent patches in the target lattice must have the same orientation. (v) Patches at a lattice position are oriented according to the (rotated) occupying species.

4.1.4 Bounded structures

Besides the torsional patches, another important difference to [30] is that the method presented here allows for bounded structures. This is achieved by adding species of type “empty” as a "shell" around the shape to ensure that empty patches remain unbound. Adding a clause $x^{B}_{0,1}$ ensures that colour 0 always binds to 1.

We then add clauses $x^{F}_{l,p,1}$ to constrain every boundary patch p at lattice position l to have the colour 1 and thereby not bind anything else. For example, in Figure 4.3, where these boundary patches are seen coloured white (and bordering an empty square), we get the following 12 clauses:

$$\begin{aligned} &x^{F}_{0,0,1} \wedge x^{F}_{0,1,1} \wedge x^{F}_{0,3,1} \wedge \\ &x^{F}_{1,0,1} \wedge x^{F}_{1,2,1} \wedge x^{F}_{1,3,1} \wedge \\ &x^{F}_{3,0,1} \wedge x^{F}_{3,1,1} \wedge x^{F}_{3,2,1} \wedge \\ &x^{F}_{4,1,1} \wedge x^{F}_{4,2,1} \wedge x^{F}_{4,3,1} \end{aligned} \tag{4.1}$$

Note that for 3D polycubes, there are six patches per species instead of the

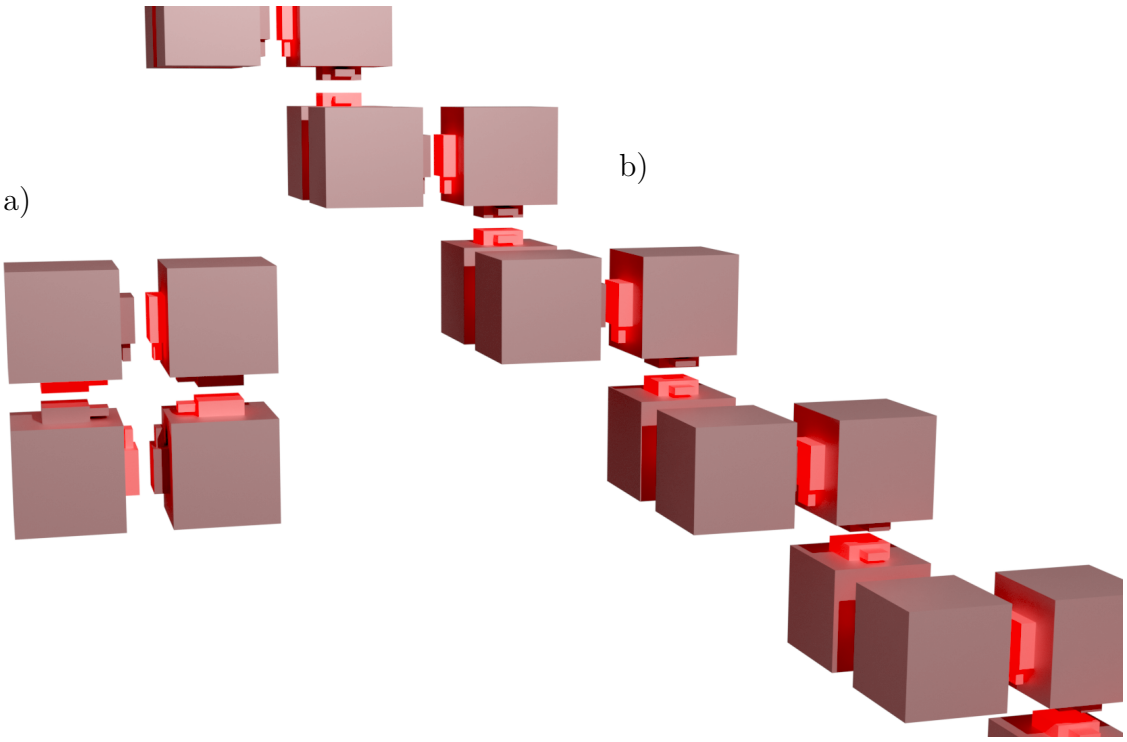

Figure 4.2: The consequences of a rotated patch. **a)** A minimal solution (one species and one colour) for a 2×2 square. **b)** The same rule as **a)**, except one patch on is rotated by $\frac{\pi}{2}$.

four seen in the 2D polyomino in Figure 4.3. This also introduces 27 possible cube rotations, compared to the 4 square rotations defined for 2D.

The topology of the shape is enforced by clause (iv), $\left(\neg x^F_{l_1,p_1,c_1} \vee \neg x^F_{l_2,p_2,c_2} \vee x^B_{c_1,c_2}\right)$ in CNF, making sure that if patch p_1 on lattice position l_1 binds to p_2 on lattice position l_2, their colours are compatible. Similarly, clause (ix) ensures that the patches have the same orientation.

4.1.5 Interaction matrix

Compared to [30], the interaction matrix is by default fixed. Thus, the $x^B_{c_i,c_j}$ variable has fixed values and we only need to extract the values of $x^F_{l,p,c}$ and $x^O_{s,p,o}$ to construct the assembly rule. Note, however, that it is still possible to re-enable a variable interaction matrix, something which could prove useful for some shapes where, for example, self-complementary patches would result in a lower complexity.

Compared to the interaction matrix convention used in Chapter 3, where each

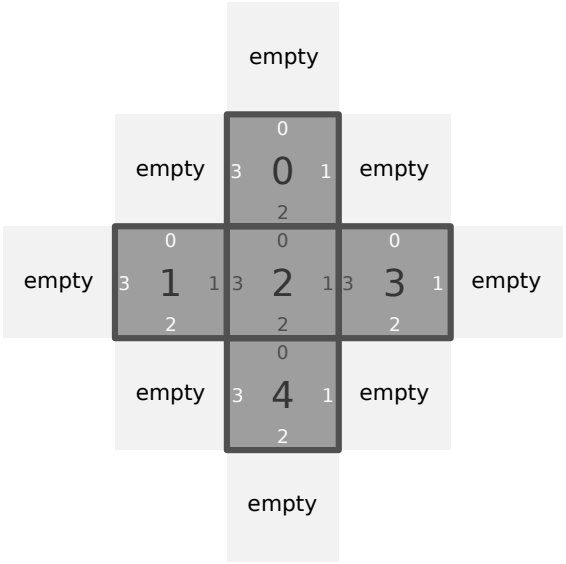

Figure 4.3: Bounded shape topology for satisfiability solving. Patches at the boundary of the shape (white) are constrained to only bind to "empty". 3D shapes are specified the same way, but with six patches per species.

colour c binds to $-c$, the colour values in the SAT solver remain unsigned and instead pair each even colour c to the odd $c + 1$. The colour pairs are mapped back to the polycube convention when obtaining the solution.

4.1.6 Assembly determinism

Even if the SAT solver determines that a solution exists, it is still possible that the rule we get can also assemble into other shapes. Recall, once more, the "giraffe duck" shape from Figure 3.2.b). If we solved for the shape with two neck cubes, the non-deterministic rule shown would be a perfectly valid solution according to the SAT solver, even though it can also produce giraffe ducks with any other neck length.

Because of this, we use the stochastic assembler to verify that the rule assembles into the correct shape every time. Each potential rule is evaluated a large number of times (by default 100), calculating an assembly ratio. If the ratio is 1, the rule is considered bounded and deterministic and, as such, a valid solution.

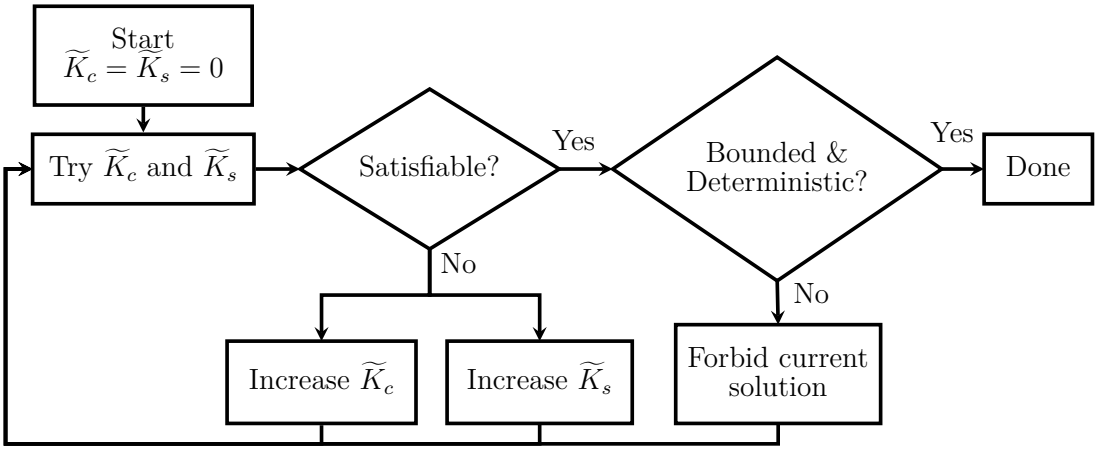

Figure 4.4: Algorithm for finding the minimal solution using SAT. Even if a solution is found to be satisfiable it might not assemble correctly every time. Additional solutions for a given $\widetilde{K}_c$ and $\widetilde{K}_s$ are found by explicitly forbidding the current solution. Alternatively, it is possible to use a solver like relsat to obtain multiple solutions.

4.2 Finding the minimal assembly rule

By iteratively ruling out lower values of $\widetilde{K}_s$ and $\widetilde{K}_c$, a minimal solution can be found, as detailed in Figure 4.4. It is also possible to generate and compare alternative solutions of varying complexity. The exploration of the solution landscape can also be done in parallel, with each combination of $\widetilde{K}_s$ and $\widetilde{K}_c$ explored concurrently.

4.3 Simplification by substitution

An alternative and complementary approach is to substitute species that are similar, removing duplicates. Starting from a fully addressable solution, any pair of species s_1 and s_2 with the same configuration of patches are tested. If we can remove s_1 and replace the patches complementary to it with ones complementary to the patches of s_2 and still get the correct output shape, we have successfully simplified the rule. This substitution continues until all species pairs have been tried.

4.4 Example solves

This section presents a set of shapes solved to demonstrate the method.

Robot

Figure 4.5: Solution landscape for assembling a polycube "robot" shape.

Figure 4.6: Solution landscape for assembling a polycube "swan" shape.

Swan

Figure 4.7: Solution landscape for assembling a polyomino letter "J" shape.

Polyomino J

Hollow cube Figure 4.8 shows the solution landscape for a hollow $3 \times 3 \times 3$ cube.

Solid cube Figure 4.8 shows the solution landscape for a solid $3 \times 3 \times 3$ cube.

4.5 Patchy particle simulation

While the stochastic assembler works well to test determinism, a more realistic assembly can be achieved through patchy particle simulation, as introduced in Section 2.2.4.

The oxDNA patchy particle simulator used is a version with torsional interaction enabled, as detailed in Section A.2.3. Since there are different *narrow types* available, corresponding to slightly different patch interaction potential widths, we start by simulating the structure stability for each potential and for a range of different temperatures.

Next, we then try to simulate and compare the assemblies of the fully adressable and minimal solutions to the shapes introduced in Section 4.4.

A simple measure available from the simulations is the potential energy in the system, where a sudden drop over time corresponds to a nucleation event.

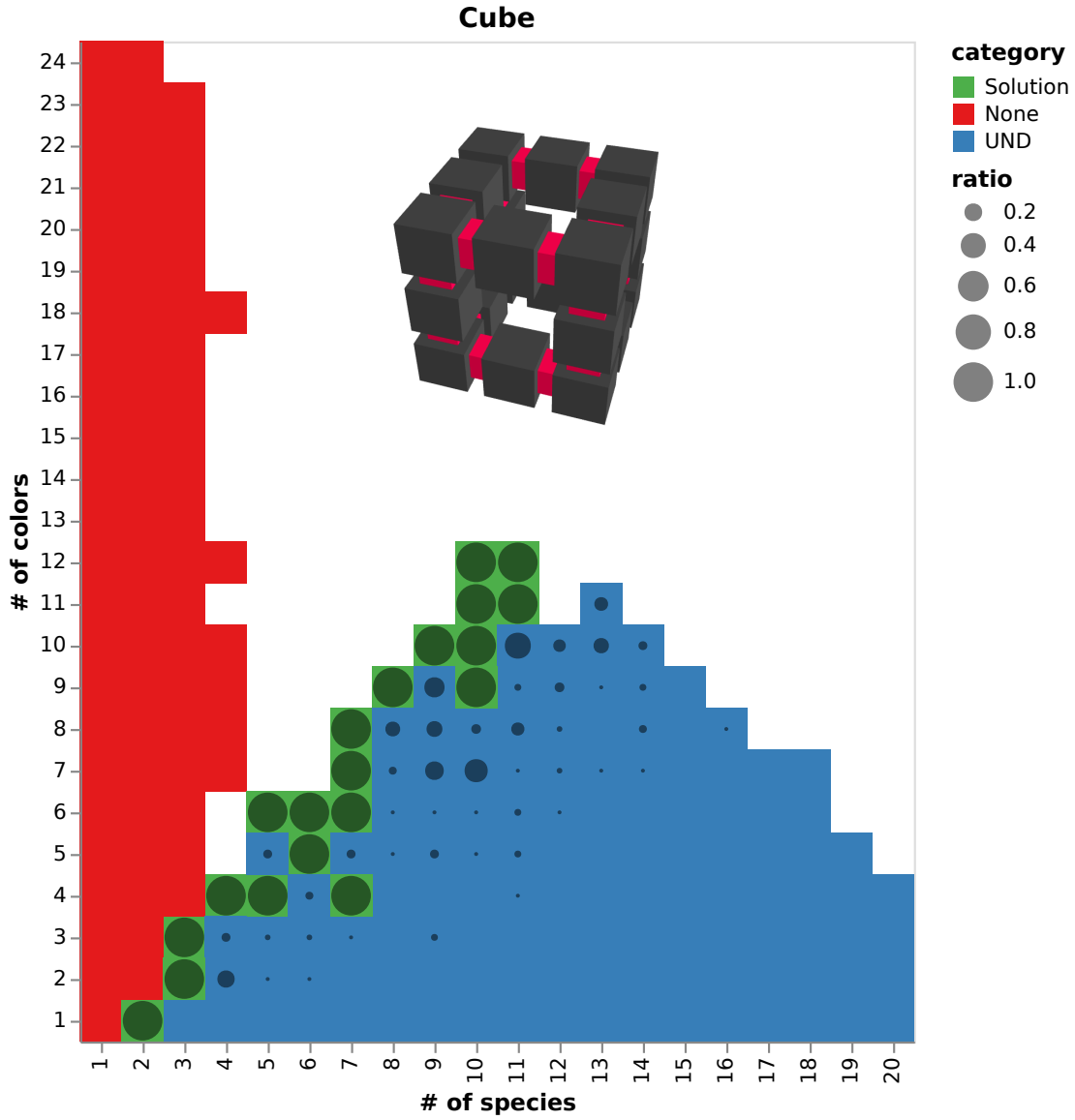

Figure 4.8: Solution landscape for assembling a hollow $3 \times 3 \times 3$ cube.

Figure 4.9: Solution landscape for assembling a polycube hollow $3 \times 3 \times 3$ cube.

4.6 Multifarious assemblies

Another feature with the SAT solver approach is the ability to design *multifarious* assemblies, that is, rules that can assemble into more than one shape. This is simply done by defining multiple distinct shapes next to each other as input to the solver.

Figure 4.13 shows a solution landscape for two different rectangle shapes. The fully adressable solution (in the upper-right corner) has no shared species between

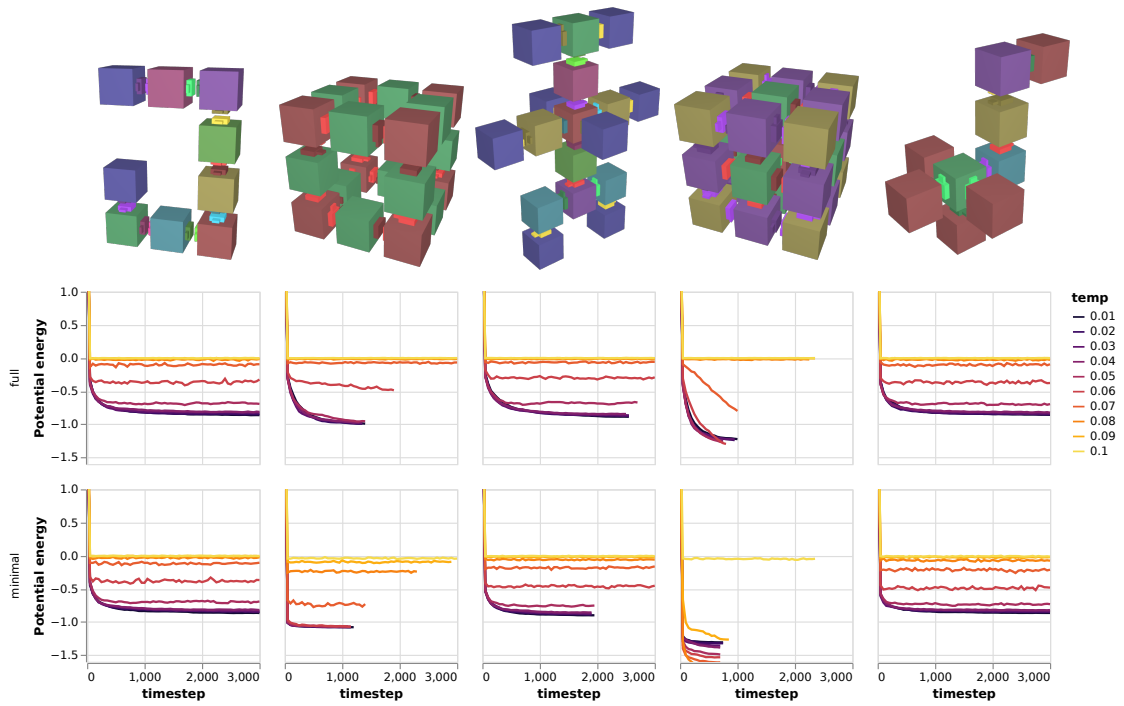

Figure 4.10: Potential energy over time in patchy particle simulations. Each simulation is done using the narrow type 0 potential at a 0.1 particle density.

the two shapes, while a simpler solution, such as: 0d018600000005018600000009050200000008d0102880000 with $\widetilde{K}_s = 4$, $\widetilde{K}_c = 3$, share three out of four species.

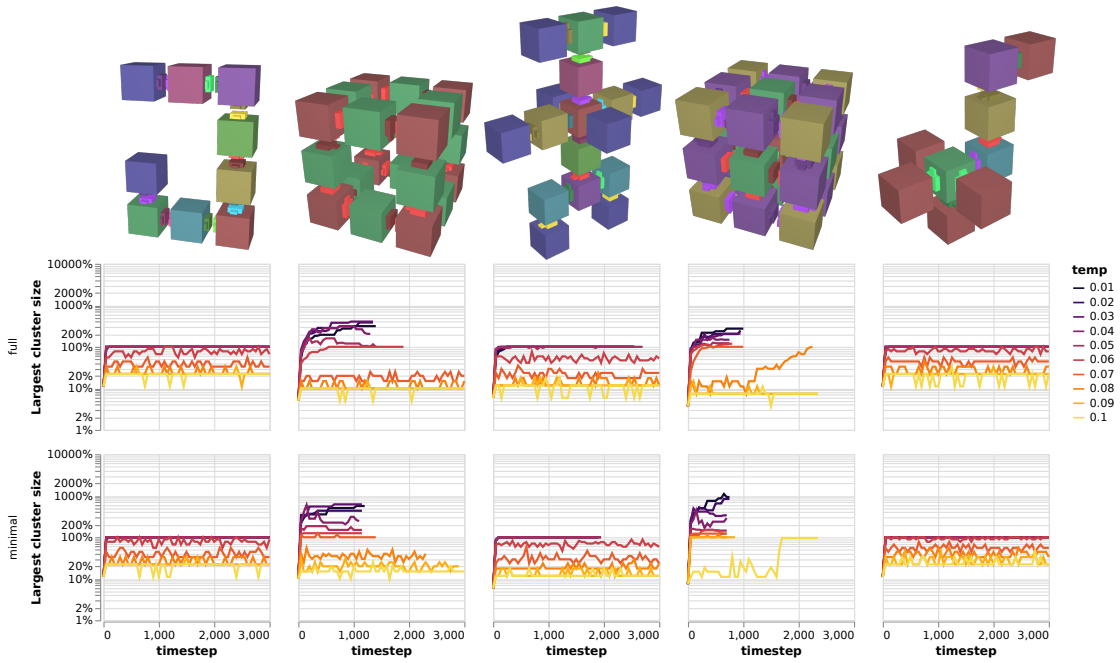

Figure 4.11: Largest cluster size over time for patchy particle simulations. Each simulation is done using the narrow type 0 potential at a 0.1 particle density.

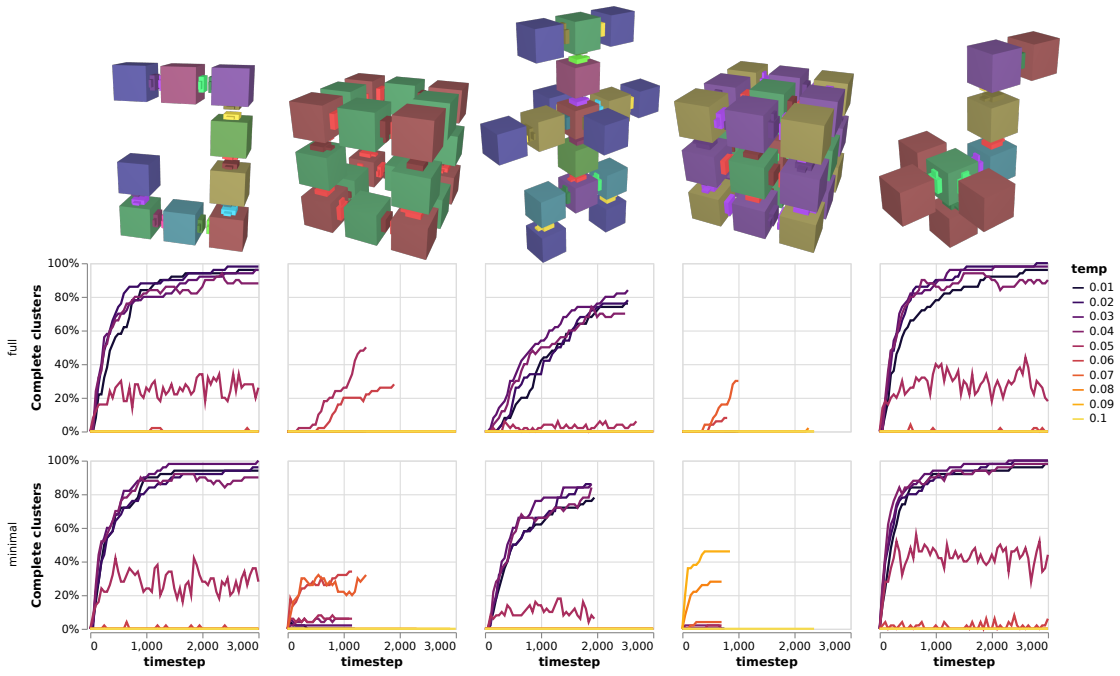

Figure 4.12: Assembly yield over time for patchy particle simulations. Each simulation is done using the narrow type 0 potential at a 0.1 particle density.

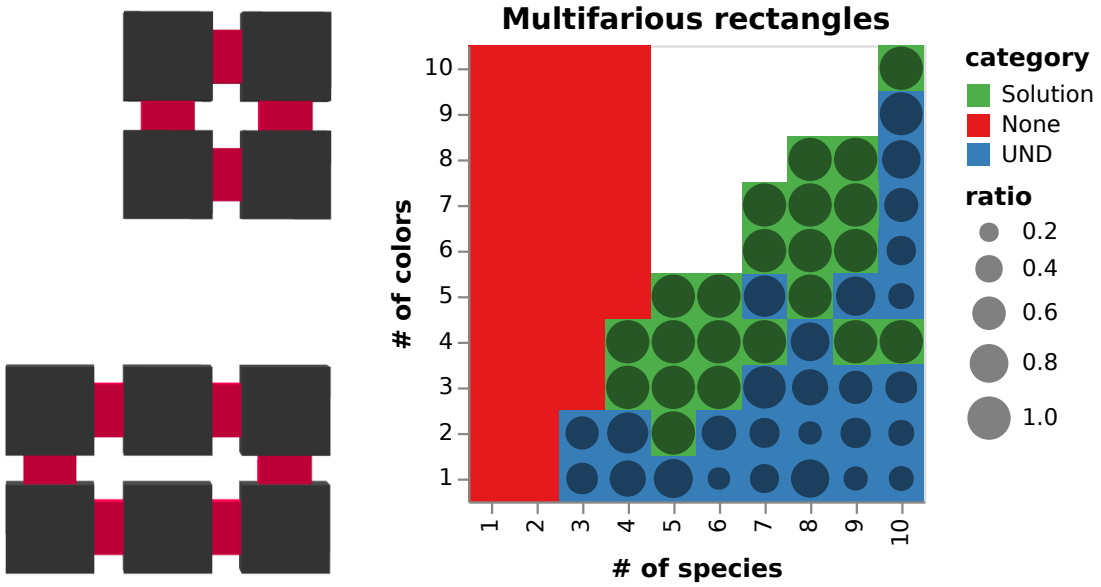

Figure 4.13: Multifarious assembly of rectangles. Solution landscape (right) for the assembly of two different rectangular shapes (left).

5

An introduction to tools for the design and simulation of nucleic acid structures

Contents

While previous chapters have covered modular self-assembly on a very abstract level, approximating the modules as simple cubes or patchy particles, this chapter will introduce tools and methods for designing and simulating individual structures or modules folded using DNA (or RNA).

The following sections will cover a selection of practical design and simulation tools that have been developed over the years, providing context for the presentation of my contributions to the *oxView* tool in Chapter 6.

5.1 Design tools

Designing a DNA origami structure by hand would be very laborious for anything but the most simple design. As such, a host of computer-aided design tools have been introduced over the years to make things easier. This section will cover some of the more common examples.

5.1.1 Lattice-based design tools

The caDNAno design tool [35] and the web-based scadnano [36] it inspired, allows the user to design DNA origami on a lattice of parallel helices.

caDNAno

CaDNAno [35] was introduced in 2009 as a way to simplify 2D and 3D DNA origami designs. It has a graphical user interface with multiple panels, seen in Figure 5.1. In the slice panel, the designer can place virtual helices on a lattice (either hexagonal or square), seen in the leftmost panel of the figure. The helices can then be filled in with strands and connected using crossovers in a path panel, seen in the middle of the figure.

Finally, caDNAno is also available as a plugin to the Autodesk Maya software, which enables a 3D visualisation of the design as seen in the render panel to the right in Figure 5.1. However, caDNAno does not support Maya versions after 2015 [37].

Scadnano

Scadnano [36] (scriptable caDNAno) is a relatively new design tool, independent from but inspired by caDNAno version 2. The main difference is that Scadnano is entirely web-based (thus not requiring any installation). The python code base is also designed to make it easier to write scripts generating DNA designs. Scadnano can be found at `https://scadnano.org/`.

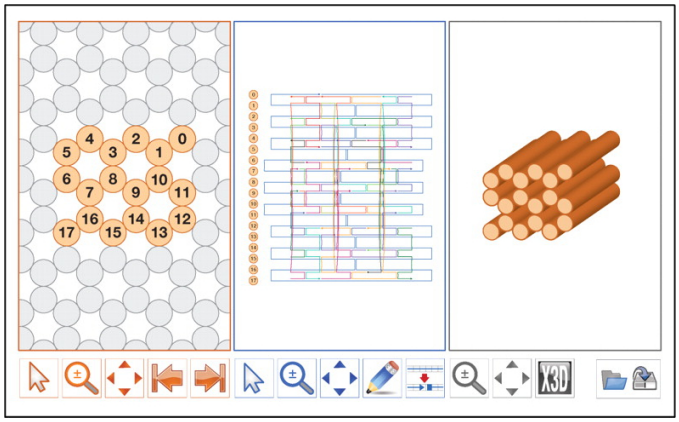

Figure 5.1: The caDNAno design interface, adapted from [35]. The slice panel (left) shows helices as circles on a lattice, while the path panel (centre) shows individual strands from a flatened side view of the helices. The rightmost panel shows a 3D visualisation of the design.

5.1.2 Top-down shape converters

While tools like caDNAno simplify bottoms-up design, where the user builds structures from individual strands and nucleotides, a top-down tool can take a polyhedral target shape as input and provide a suitable origami design as output.

BSCOR

In 2015, Benson et al. published a method for converting arbitrary mesh designs into a DNA origami mesh [38]. Figure 5.2 shows a set of example polyhedral shapes, with the designed shape in **a)**, the output DNA design in **b)**, and microscopy characterisations in **c)**-d). A follow-up paper in 2016 also introduced the ability to design flat-sheet meshes [39]. BSCOR uses single DNA duplex edges, with double edges added whenever topologically necessary.

ATHENA

ATHENA is a recently published tool for automatic design wireframe origami shapes. As seen in Figure 5.3, earlier software such as PERDIX, METIS, DAEDALUS and TALOS has facilitated design for 2D and 3D wireframes using different edge designs but ATHENA aims to bring them all together as a single package.

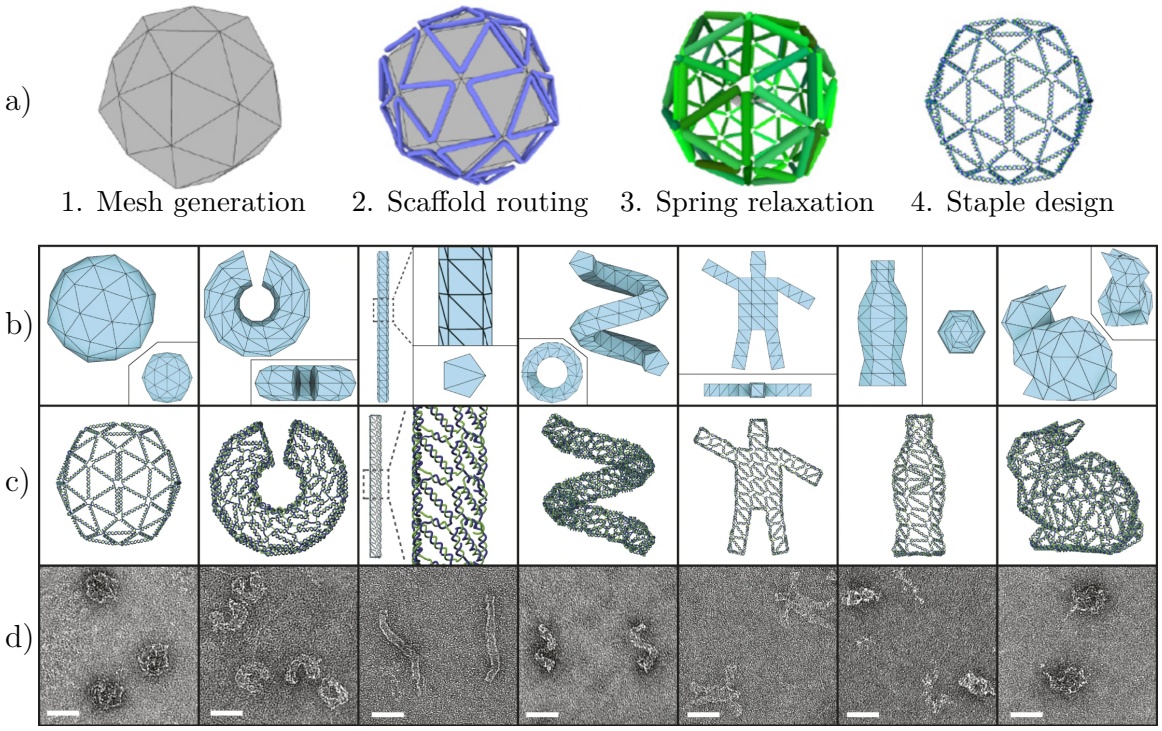

Figure 5.2: 3D meshes rendered in DNA origami using BSCOR. Adapted from [38] and [40]. **a)** Automated design process, where a scaffold is routed onto a mesh, each helix is relaxed using spring forces, and staple strands are added. **b)** Examples of initial meshes. **b)** Completed DNA designs with strands rendered as tubes. **c)** Negative-stain dry-state TEM micrographs of each design.

5.1.3 Free-form or hybrid tools

The final category of design tools is either free-form, where designs are drawn without a lattice, or hybrid tools combining both lattice and free-form design.

Tiamat

Tiamat is an early free-form design tool introduced in 2009 [42], running on Microsoft Windows. See Figure 5.4 for a screenshot of the user interface of Tiamat 2, where a DNA tetrahedron is being designed. Tiamat can also import DNA from PDB files.

vHelix

The free-form tool vHelix [38] is a plugin for the Autodesk Maya software and was developed together with the BSCOR toolkit. Users can import the "rpoly" wireframe result from BSCOR, create designs from scratch, or import caDNAno

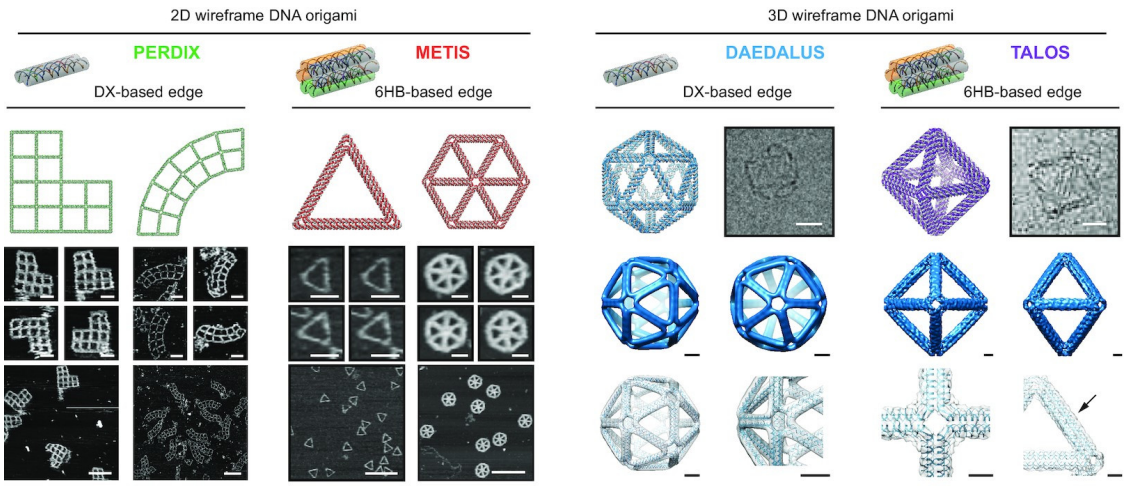

Figure 5.3: Automatic wireframe origami shapes using ATHENA. Adapted from [41]. ATHENA includes the previous design packages PERDIX, METIS DAEDALUS and TALOS for 2D and 3D wireframe design, using double crossover (DX) and six-helix bundle (6HB) edges respectively.

files. It is also possible to export oxDNA simulation files. An example of the vHelix interface is shown in Figure 5.5.

Adenita

Adenita [44] is a free-form editing tool developed as a plugin to the SAMSON toolkit. As seen in Figure 5.6.a), the design can be visualised and edited at multiple levels of abstraction, from an all-atom representation, through nucleotides and strands to cylinders representing entire helices. Figure 5.6.b) shows some of the available editing and visualisation tools. Note, for example, the wireframe creation tool based on the Daedalus algorithm. Adenita can also load caDNAno files and export oxDNA simulation files.

Like vHelix (Section 5.1.3), Adenita is tied to the commersial editor it is a plugin to.

MagicDNA

MagicDNA [45], as seen in Figure 5.7, has a computer-aided design workflow where geometry can be specified from helix cross-sections or imported from a part library, then assembled into an integrated structure (using multiple scaffolds if necessary).

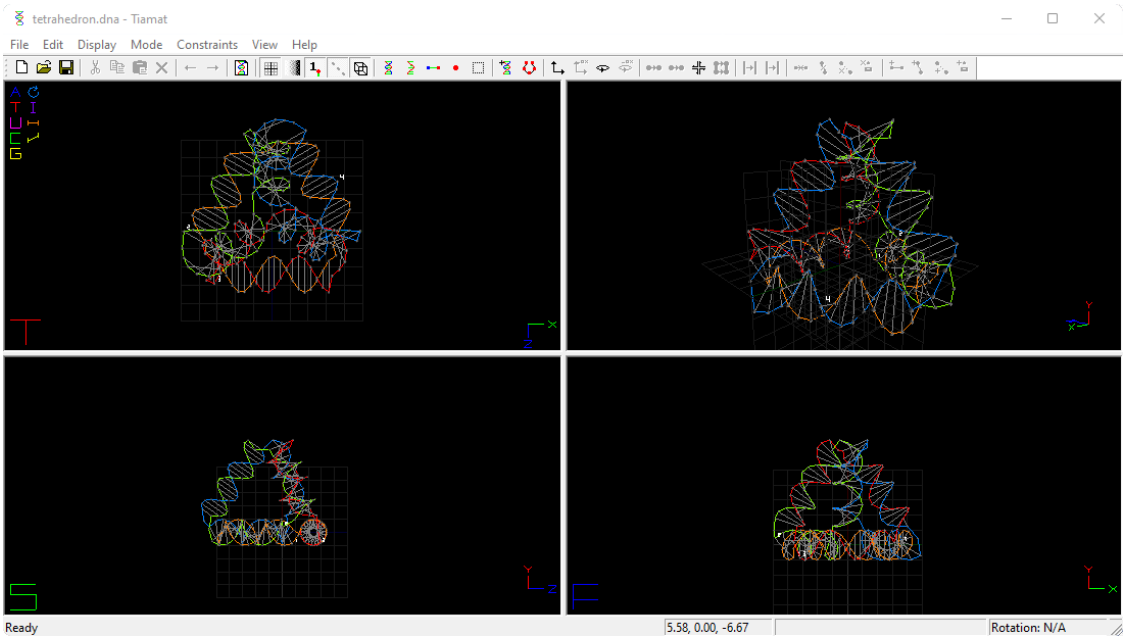

Figure 5.4: A screenshot of the Tiamat [42] (v2) user interface. The loaded tetrahedron design is from the Yan Lab resources page [43]

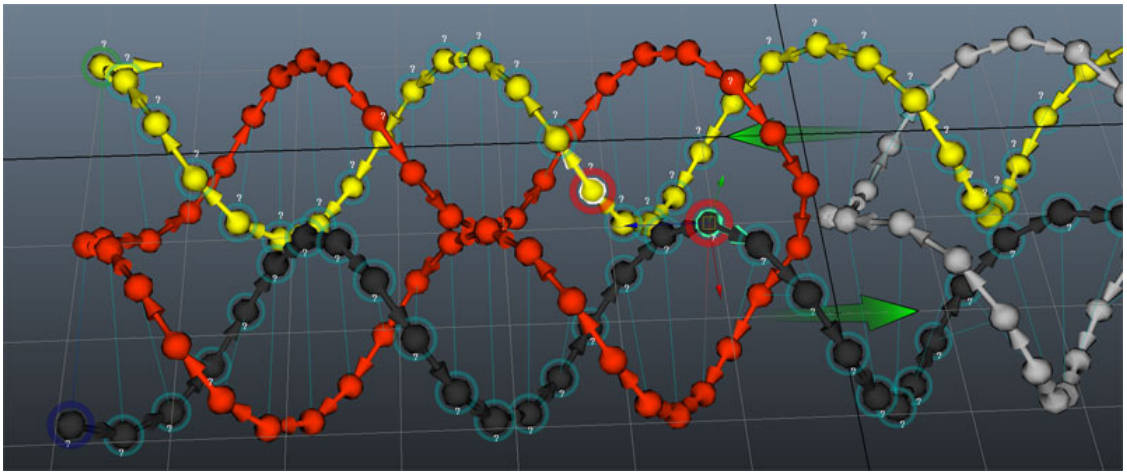

Figure 5.5: Free-form editing in vHelix. Image is from the vHelix website `http://www.vhelix.net/` [40].

MagicDNA is built as an application for the MATLAB software, so (like the previous two tools) installation requires a commersial third-party tool.

OxView

The oxView application was developed as part of this thesis project and will be described more in Chapter 6.

a)

b)

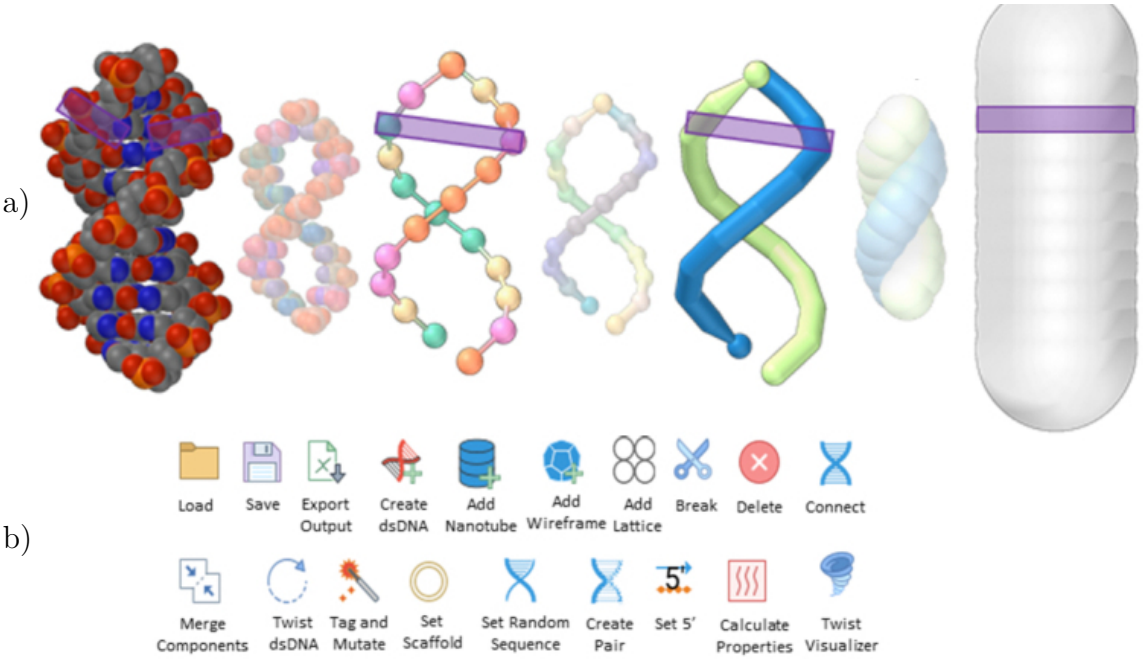

Figure 5.6: Adenita, adapted from [44]. **a)** A DNA double-helix visualised at different abstraction levels, with a purple band highlihghting a specific base-pair at all the four main levels. **b)** Available editing and visualisation tools in Adenita.

5.2 Simulation models

Simulating a structure can provide insight to understand experimental results but can also guide decisions at the design stage. More coarse-grained models tend to run faster, but may lose some accuracy compared to models with more detail. This section covers simulation models at an increasing level of coarse-graining, from individual atoms to cylindrical helices.

5.2.1 All-atom simulation

Simulation tools such as NAMD [46], use force fields such as AMBER [47] and CHARMM [48] that model interactions between individual atoms. While it is possible to perform atomistic simulations of large DNA origami structures [49], the simulations take a long time to run, and it is unknown how well the models represent DNA thermodynamics [50].

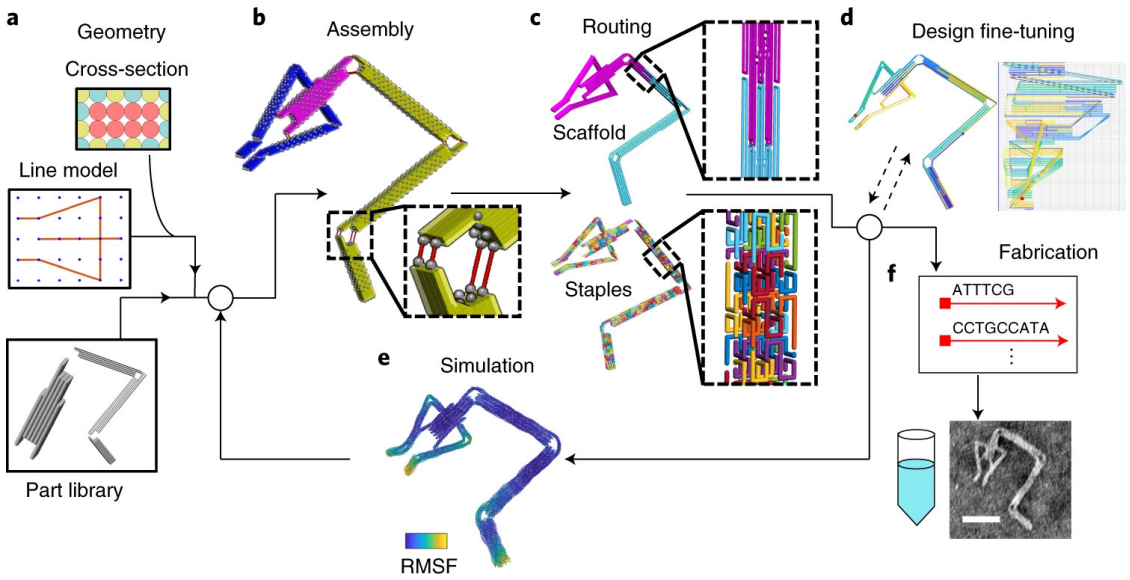

Figure 5.7: MagicDNA adapted from [45].

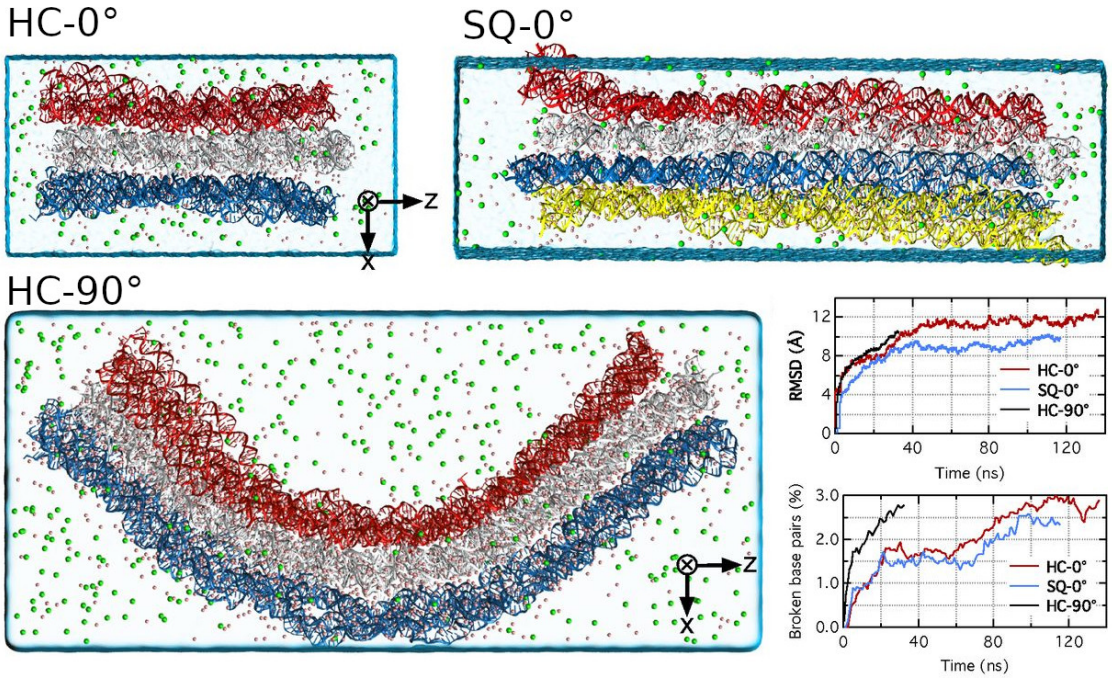

Figure 5.8: All-atom simulation

5.2.2 oxDNA/RNA

In 2010, a coarse-grained simulation software called oxDNA was introduced by Thomas Ouldridge [5]. It simulates DNA on the level of nucleotides and has been shown to model complex origami devices with a generally good agreement with

experimental data [51]. In 2014, the DNA model was extended to include RNA by Petr Šulc [52], showing its ability to model a set of common RNA motifs.

Molecular Dynamics (MD) and Monte Carlo (MC) simulation techiniques.

OxDNA was joined by cogli1 for trajectory visualisation.

While oxDNA can be very useful for modelling a structure, it has traditionally not been very accessible for experimentalists.

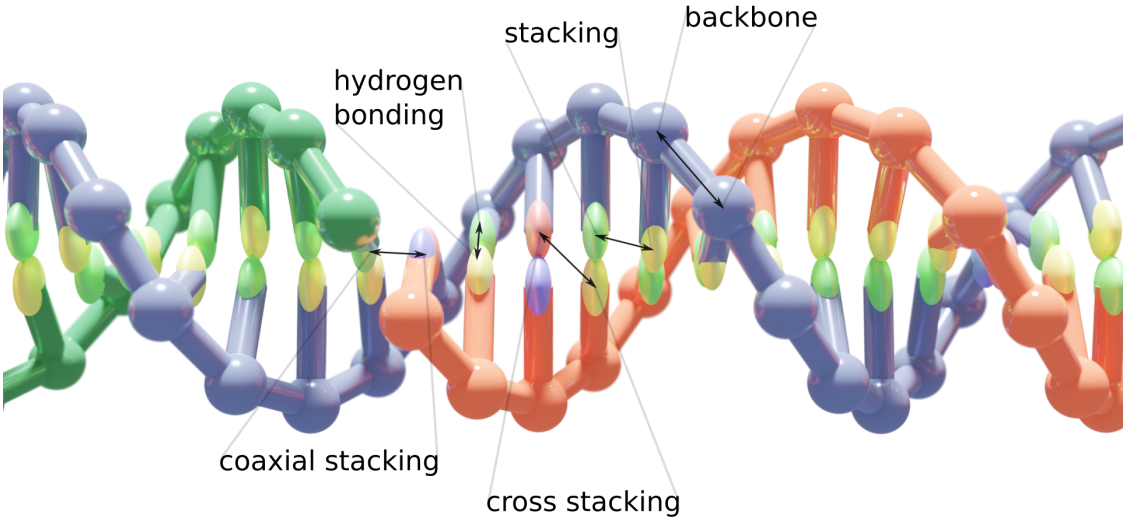

Figure 5.9: The oxDNA model

5.2.3 mrDNA

The mrDNA simulation model is a multi-resultion model representing a user-defined number of base-pairs as a rigid-body bead.

Since mrDNA is implemented in Python and has a spline-based helix representation, users can write scripts to edit the structure, translating and rotating parts before starting the simulation. Thus, topological issues or over-stretched bonds can, with some skill, be resolved even before starting to simulate.

I did, based on this, also create a rudimentary interactive editor interface to mrdna, but it would need a lot of refinement to be externally usable. More importantly, the oxView editor described in Chapter 6 can now easily resolve these issues.

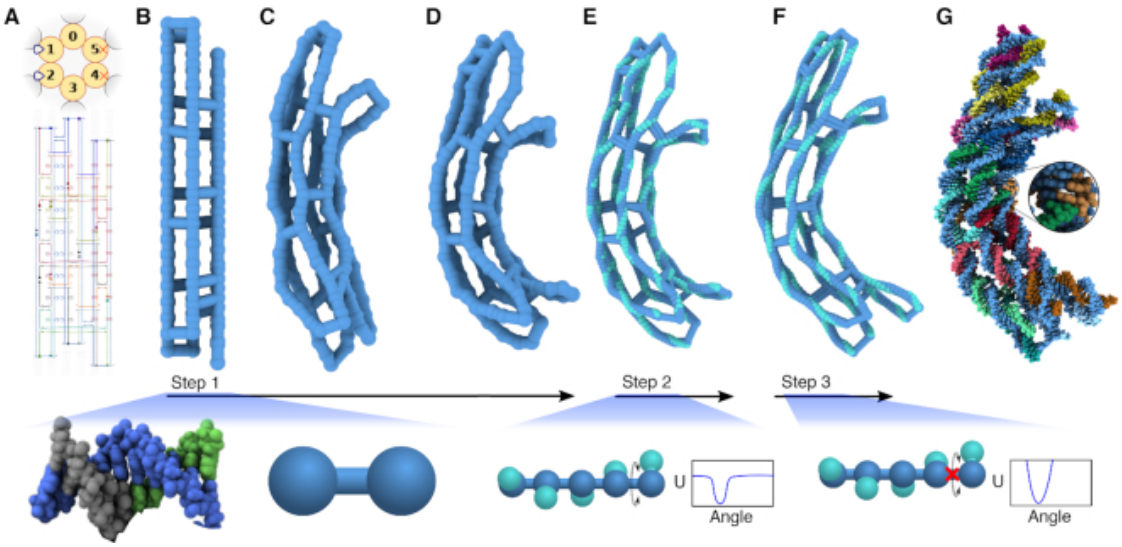

Figure 5.10: MrDNA

A selection of the DNA designs I have relaxed are shown in Figure 5.11. The first two examples, adopted from [53] and [54] and illustrated in Figure 5.11.a and 5.11.b, are both quite straightforward to relax in oxDNA, although the relaxation is much faster using mrdna.

The tensegrity kite structure, adopted from [55] is harder to relax since, as seen in the first image in Figure 5.11.c, the two helix bundles are drawn parallel to each other in caDNAno. Given enough time to relax, they should still become orthogonal, but a much more efficient way is to write a mrdna script to rotate one helix bundle so that it is orthogonal from the start, as seen in the middle image of 5.11.c. The remaining overstretched bonds are then quickly relaxed using mrdna.

Finally, the Möbius strip, adopted from [56], is particularly tricky to relax, since the caDNAno design have all helices drawn in the same plane, with bonds from each end stretching through the whole structure and intersecting at a single point, as can be seen in the first image of Figure 5.11.d. With some help from Chris Maffeo, however, I was able to use a mrdna script to edit the structure into a configuration much easier to relax, as seen in the second image of Figure 5.11.d. Since the caDNAno design does not make it clear if the Möbius strip should be left-handed or right-handed, this is also decided in the script; changing the

rotational direction will produce a mirrored version of the structure, as seen in the third image of Figure 5.11.d.

5.2.4 Cando

Cando is a finite element modelling framework [57, 58] available through a web server at `https://cando-dna-origami.org`. DNA double helices are modelled as elastic rods (connected by rigid crossovers) that stretch, twist and bend in line with experimental measurements. See Figure 5.12 for a set of example structures designed in caDNAno and simulated in Cando.

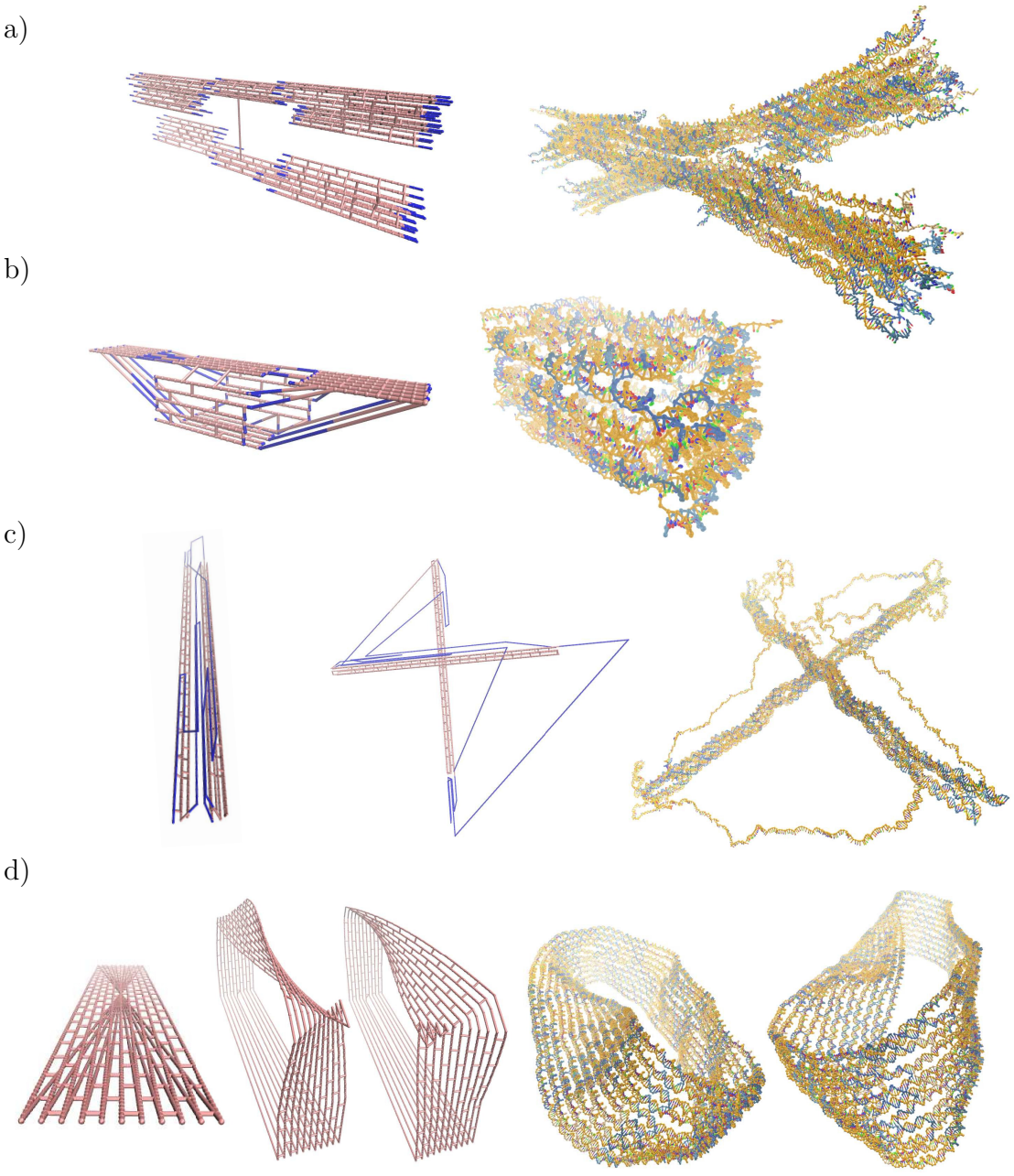

Figure 5.11: Relaxation results for various DNA designs. Each row depicts a new design, with the left-hand side showing the structure as it was drawn in caDNAno (and parsed by mrdna), while the right-hand side is the relaxed structure in oxDNA. Intermediate images are edits done in mrdna. While the switch design [53] in **a)** and the small DNA origami box [54] in **b)** relaxed without any required editing, the tensegrity kite structure [55] in **c)** and the Möbius strip [56] in **d)** benefited greatly from moving selected helices to a position off the lattice before starting the simulation.

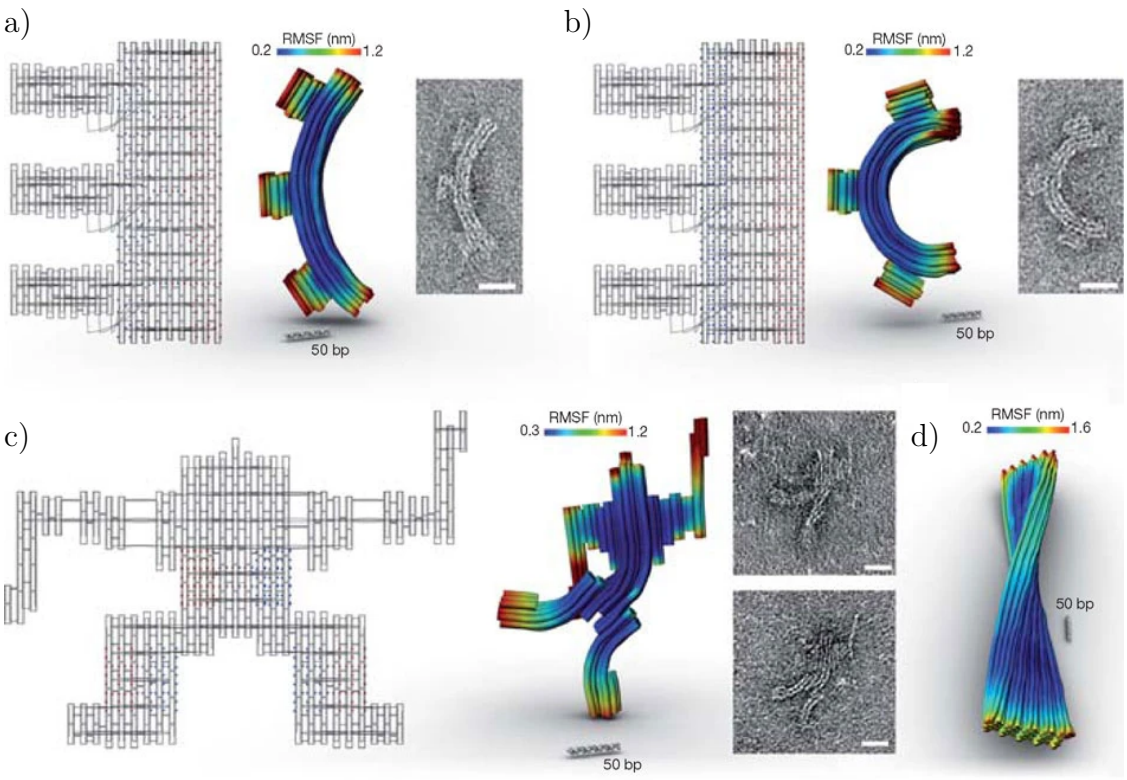

Figure 5.12: Cando simulation results. Adapted from [57]. CaDNAno design diagrams, Cando structure and flexibility prediction, and negative-stain TEM micrographs **a)** 90 degree gear. **b)** 180 degree gear. **c)** "Robot" design **d)** 60-helix bundle.

6

Structure design and analysis in oxView

Contents

This chapter contains my results on how to design, simulate and analyse DNA and RNA nanostructures.

As I started this DPhil, I was tasked with the issue of converting origami designs created in caDNAno (see Section 5.1.1) so that they could be correctly simulated in oxDNA (see Section 5.2.2). I soon started collaborating with Hannah Fowler from the Doye group at Theoretical Chemistry, who simulated an extensive collection of old DNA designs. This was a good opportunity to investigate why some structures

were more problematic to convert than others.

At this time, the available method for converting a caDNAno design into the oxDNA format was to use an old python script included in the UTILS directory of the oxDNA repository. However, it was not easy to use, and it failed for many structures. At the end of 2018, the tacoxDNA webserver [59] was launched, updating the conversion script and making it more accessible.

Still, since caDNAno structures are drawn on a lattice, where all helices have to be parallel to each other, the resulting oxDNA configurations often had unnaturally extended backbone bonds, requiring time-consuming relaxation.

During a secondment within the Šulc group at Arizona State University in 2019, I contributed to the development of a web-based oxDNA viewer called oxView [60]. Among the main early features that I added to oxView was a cluster-level rigid-body dynamics option (detailed in Section 6.2) that in many cases speed up the relaxation by orders of magnitude compared to oxDNA relaxation alone. Since then, I have collaborated with the Šulc group to add more features and to make the tool more accessible as a visualiser and editor. For our second oxView publication, I rewrote the main parts of the taxoxDNA codebase into TypeScript, resulting in the *taxoxdna.js* library (`https://github.com/Akodiat/tacoxdna.js`) which oxView now uses to import various standard design formats (including caDNAno) automatically.

This section will present my own contributions to oxView, unless otherwise stated. However, I also want to acknowledge the work done by Erik Poppleton and Michael Matthies; without them, this tool would not exist. Michael was the original oxView developer and has done great work in supporting live oxDNA relaxations through the *ox-serve* webserver. Erik enabled oxView to render and analyse systems with over a million nucleotides and has created a large set of useful analysis scripts.

6.1 Importing designs

There are a lot of different formats available for DNA origami design; some of the main design tools producing them are covered in Section 5.1. This section will describe how to use oxView to import different designs.

6.1.1 Basic import

Thanks to the *tacoxdna.js* library, importing designs is now generally straightforward. Any loaded structure can then be exported for oxDNA simulation, as described in Section 6.5. The formats listed below can all be imported by simply clicking the *Import* button in oxview. However, some additional formats still require the use of the external Tacoxdna webserver [61].

Importing caDNAno files

JSON files created using caDNAno (described in Section 5.1.1) can now be imported directly into oxView. See the import dialog shown in Figure 6.1.a). First, select the file to import and make sure to also select *caDNAno* as the file format. Next, choose the correct lattice-type; either *Square* or *Hexagonal*. Optionally, input a sequence to assign to the origami scaffold (which will otherwise be random).

Another option is to use the tacoxdna webservice to convert the caDNAno design into oxDNA files and then load those into oxView. However, designs loaded directly into oxView have the benefit of including correct colouring, clustering and base-pairing information, which would otherwise have been lost.

Importing rpoly files

Rpoly files are the output from the BSCOR [38] tool (described in Section 5.1.2) for converting polyhedral meshes into DNA origami. Select the rpoly file to import, making sure that *rpoly* is selected as the file format. Optionally, input a sequence to assign to the origami scaffold (which will otherwise be random).

Importing Tiamat files

Select the Tiamat *.dnajson* file to import, making sure that *tiamat* is selected as file format. Binary *.dna* Tiamat files need to be reopened in Tiamat and saved to the text-based *.dnajson*. Select Tiamat version (1 or 2), then select the nucleic acid type (DNA or RNA).

By default, nucleotides without assigned base types will be given a random type. However, it is also possible to select a fixed default base.

Importing PDB files

While I did include the DNA PDB to oxDNA converter from Tacoxdna in *tacoxdna.js*, a more versatile PDB import was created by Jonah Procyk to support his ANM-oxDNA model [63]. Simply drag and drop (or load) a PDB file into an oxView window and the DNA, RNA and/or protein it contains will be automatically converted and loaded.

6.1.2 Multi-component designs

Designs spread across multiple files (or even multiple design tools) can be easily combined in oxView by simply importing them all and using the editing tools to arrange and connect them properly.

6.1.3 Far-from-physical caDNAno designs

Some structures, while converted without failure to the oxDNA format, will have a very far-from-physical configuration due to the way they are drawn in caDNAno. As mentioned above, since it is only possible to draw all helices parallel to each other (on a lattice), backbone bonds may be very elongated, creating high energies and/or topological problems.

The oxDNA software already includes relaxation procedures for such structures, bringing them together in a slow and controlled manner using a specified maximum backbone force. However, for large structures, this can take a very long time, even while using GPU simulation.

In such cases, another software I have investigated, called mrdna (Multi-resolution DNA nanotechnology), proves very useful [64]. It is a python package using ARBD (Atomic Resolution Brownian Dynamics), both being developed by Chris Maffeo, to simulate double- and single-stranded DNA structures at multiple levels of coarse-graining. Since these simulations are more coarse-grained than oxDNA, they run significantly faster. Furthermore, after I have been in contact with Chris, mrdna can now also be configured to output and simulate oxDNA

configurations as the final simulation step. Because of this, mrdna can also be useful for converting simple structures if tacoxDNA fails to parse the caDNAno file.

6.2 Rigid-body manipulation and dynamics

Rapid relaxation of converted cadnano designs

Rigid-body manipulation [65]. The translation and rotation tools in oxView allow users to select and rearrange blocks of nucleotides as rigid bodies.

Dynamics [66]

Manually, on import or through DBSCAN algorithm [67].

Clusters held together with spring forces at each shared backbone bond, with a magnitude of

$$f_{\text{spr}} = c_{spr}(l - l_r),$$

where c_{spr} is a spring constant, l is the current bond length and l_r is the relaxed bond length. To avoid overlaps, a simple linear repulsive force, of magnitude

$$f_{\text{rep}} = max\left(c_{rep}\left(1 - \frac{d}{r_a + r_b}\right), 0\right)$$

is added between the centre of each group, where c_{rep} is a repulsion constant, d is the distance between the two centres of mass, and $r_a + r_b$ is the sum of the group radii (the greatest distance they can be while still overlapping).

As seen in Figure 6.2,

6.3 Editing designs

While oxView started as a visualisation tool, it has since been extended with a number of editing features. One of the earliest was the ability to perform rigid-body manipulation of selected nucleotides by dragging them with the mouse. I contributed by adding transformation gizmos that simplify translation and rotation of selections using on-screen arrows and arcs for the user to manipulate. See Table 6.1 for a complete list of the editing tools currently available in oxView.

With the ability to create, remove, connect, and disconnect nucleotides, oxView users can now design structures from scratch. See, for example, Figure 6.3, where the DNA tetrahedron from [68] is created. The user first creates an initial helix (Figure 6.3.a) by typing a 20-base sequence and clicking the "Create" button in the "Edit" tab. Note that the "Duplex mode" need to be active in order for the complementary strand to be created automatically. Next, the user can copy and paste the helix repeatedly, using the "Translate" and "Rotate" tools to position the helices as seen in 6.3.b).

The do/undo system

6.4 Visualization options

Depending on the design, a user of oxView might want to modify the visualisation settings to make certain features of the structure more or less visible.

Centring with periodic boundary conditions A useful utility when visualising an oxDNA trajectory is to keep the structure centred at the origin, stopping it from drifting out of view. The method described in [69] was used to achieve centring while taking periodic boundary conditions into account.

In essence, for each coordinate $p_j = \left(p_x^j, p_y^j, p_z^j\right)$ in the centring set of size n, each dimension $i \in \{x, y, z\}$ gets averaged in its own variable α_i, representing its 1D interval as a 2D circle (where the circumference b_i is the bounding box side length:

$$c_i = \frac{1}{n} \sum_{j=1}^{n} \left[\cos\left(\alpha_i^j\right), \sin\left(\alpha_i^j\right)\right]$$

Where $\alpha_i^j = \frac{2\pi}{b_i} p_i^j$ is the angle on the unit circle and c_i is the average 2D position representing dimension i. Finally, the averages are converted back to cartesian coordinates:

$$cm_i = \pi + \frac{b_i}{2\pi} atan2(-c_{i,y}, -c_{i,x})$$

Change component sizes , change colours, fog, virtual reality.

Tool	Description
	**Create** a new strand from a given sequence. Select *duplex mode* to instead create a helix.
	**Copy** the selected elements (Ctrl+C).
	**Cut** the selected elements (Ctrl+X).
	**Paste** elements from clipboard (Ctrl+V to paste in original position, or Ctrl+Shift+V to paste in front of camera).
	**Delete** all currently selected elements (delete).
	**Ligate** two strands by selecting the 3' and 5' endpoint elements to connect (L).
	**Nick** a strand at the selected element (N)
	**Extend** strand from the selected element with the given sequence. Select *duplex mode* to also extend the complementary strand.
	**Insert** (add) elements within a strand after the selected element.
	**Skip** (remove) selected elements within a strand.
	**Rotate** selected elements around their center of mass (R).
	**Translate** currently selected elements (T).
	**Move to**. Move other selected elements to the position of the most recently selected element.
	**Connect 3' duplex**. Connects the 3' ends of two selected staple strands with a duplex, generated from the sequence input.
	**Connect 5' duplex**. Connects the 5' ends of two selected staple strands with a duplex, generated from the sequence input.
	**Set** the sequence of currently selected elements. Select duplex mode to also set the complementary sequence on paired elements.
	**Get**. Assigns the sequence of selected bases to the sequence input.
	**Reverse complement**. Generates the reverse complement of a provided sequence.
	**Search**. Highlights the position the provided sequence in each strand, if present.

Table 6.1: Editing tools available in oxView

6.5 Exporting designs

Once a structure has been created in or loaded into oxView, it can be exported in a variety of formats.

6.5.1 Exporting oxDNA simulation files

OxView can export topology, configuration, and external force files for oxDNA simulation. Simply click the "Export oxDNA" button in the "File" menu and select the required file types.

One important thing to note is that the oxDNA format requires nucleotides to be correctly sorted with consecutive indices. Furthermore, this sorting is done, against convention, in a 3' to 5' order. Meanwhile, to facilitate editing, oxView nucleotides keep their indices even if the topology changes. Thus, nucleotides indices may be reassigned on oxDNA export, so it is important to export all files (topology, configuration, and forces) if the design has been edited.

6.5.2 Exporting other 3D formats

File formats such as glTF and STL contain geometrical information that can be 3D printed or imported into other 3D software such as Blender. OxView exports the scene as it is at the moment of export (at the current frame if a simulation trajectory is loaded), so it is important to configure intended component sizes and colours beforehand.

STL is an old and common standard for 3D shapes, containing only vertex coordinates (no colours). It is a popular input for 3D printing, but the file size is relatively large.

The glTF format (or glb if binary) is a modern standard for 3D scenes, storing geometry, hierarchy, and even material properties.

6.5.3 Exporting sequence files

Strand sequences can be exported as standard CSV files using the "Sequence file" export button in the "File" tab. The designed sequences can then be ordered and assembled experimentally.

6.5.4 Saving image files

It is possible to save an image of the current oxView view by simply clicking the "Save image" button in the "File" tab. This is a preferred option over a screenshot for two reasons. First, it is possible to increase the resolution by a scaling factor found in the "Image size" dropdown (rescale the browser window to change the aspect ratio of the image). Secondly, the background of the saved image will be transparent, simplifying further editing and composition.

For cover art and photo-realistic renders, it is also possible to export and load a glTF file into (for example) Blender, as seen in Figure 6.5. Note that in current Blender versions (2.9), large structures take a long time to import. So, make sure to disable any components not needed before export.

6.5.5 Creating videos

One of the earlier features I implemented in oxView was the ability to create videos from oxDNA simulation trajectories. It is also possible to create lemniscate videos of single configurations, where the camera moves around the structure in a lemniscate-shaped loop. The video export uses the `CCapture.js` library, grabbing the `Three.js` canvas and outputting either "webm" or "gif" animations, or each frame as separate "jpg" or "png" images.

Click "Create video" in the "File" menu, choose video type (trajectory or lemniscate), file format, and frame rate. For lemniscate videos, it is also possible to set a video duration.

For trajectory videos, the camera can be manually moved while the video is being recorded, thus showing the simulation from different angles.

Another option for video creation is to follow the steps described in Section 6.5.4. When the structure is loaded into Blender, the camera can be animated (or the design rotated) using keyframes to render high-quality videos. To render an oxDNA simulation, use the "traj2blender.py" script, found at `https://github.com/Akodiat/traj2blender`, to automatically load keyframes corresponding to simulation steps.

6.6 Converting RNA origami designs

During my secondment at the Andersen lab in Aarhus, I worked with converting RNA structures designed using their ASCII-based blueprint format into oxRNA simulation files. Examples of converted structures are shown in Figure 6.6. The Andersen lab already has scripts, soon to be published, for parsing their blueprint files and building the corresponding PDB structures (the first two columns of Figure 6.6). However, the resulting PDB files are not relaxed and would take a long time to relax using all-atom simulation. As such, I modified the tacoxDNA [59] PDB parser to enable PDB-to-oxRNA conversion. I have contacted Lorenzo to let him know about these additions, and hopefully, they will be included in a future version of tacoxDNA. One issue I noticed with the conversion was that the asterisk (*) and prime (') characters were used interchangeably in the RNA motif library to name atoms of the sugar group, causing problems with the tacoxDNA script, which only recognises the prime. This has already been fixed in tacoxDNA. The third column of Figure 6.6 shows the structures relaxed and simulated in oxRNA, some significantly different from the previously available PDB models in the second column.

a)

b)

c)

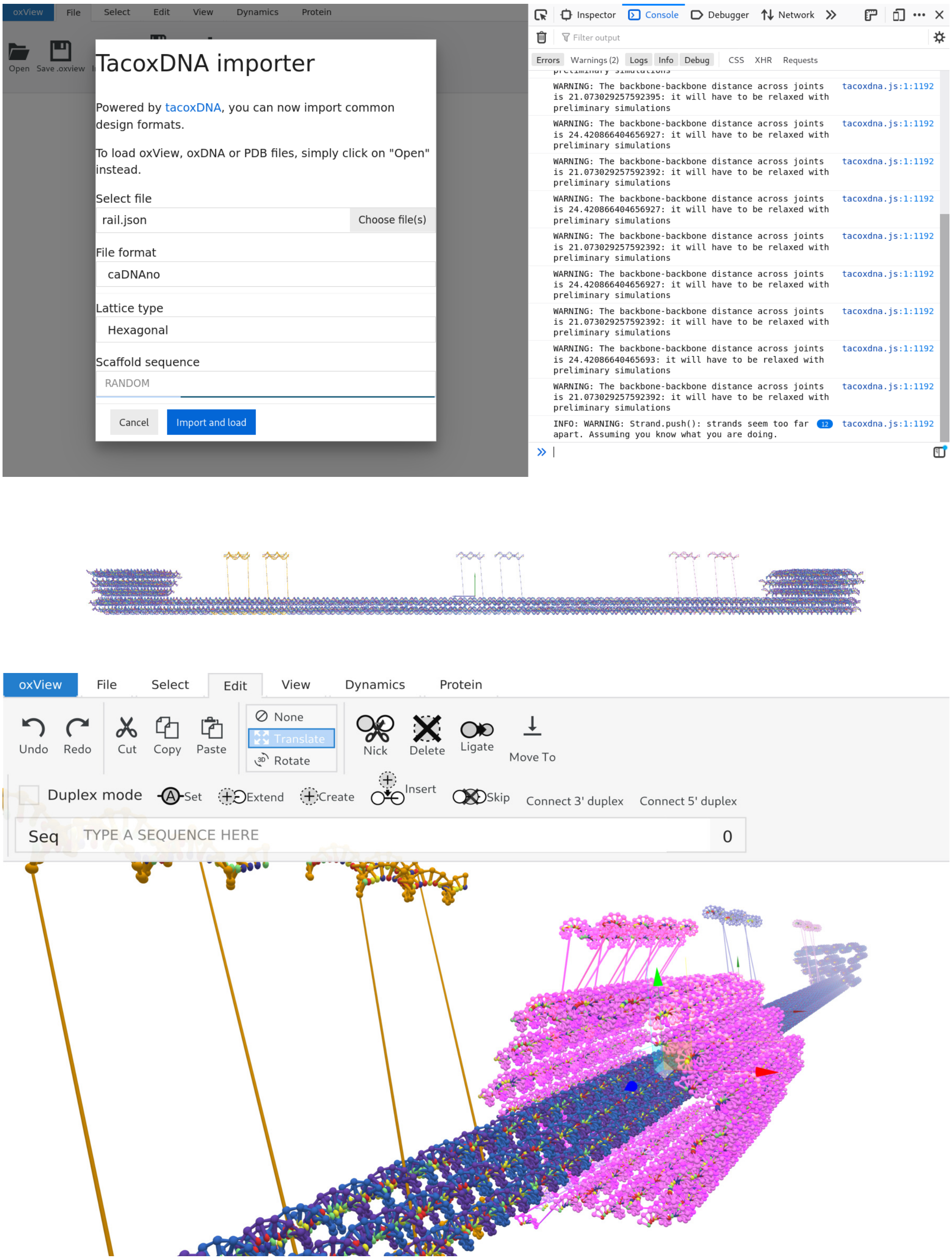

Figure 6.1: Importing caDNAno structues into oxView. **a)** The tacoxdna.js library import dialog in oxView (left), seen here importing a caDNAno design. Note the web browser console shown to the right, where any additional output from the import is written. **b)** Imported linear actuator rail design from [62]. **c)** Complete linear actuator structure from [62] assembled in oxView, after also importing the slider caDNAno design.

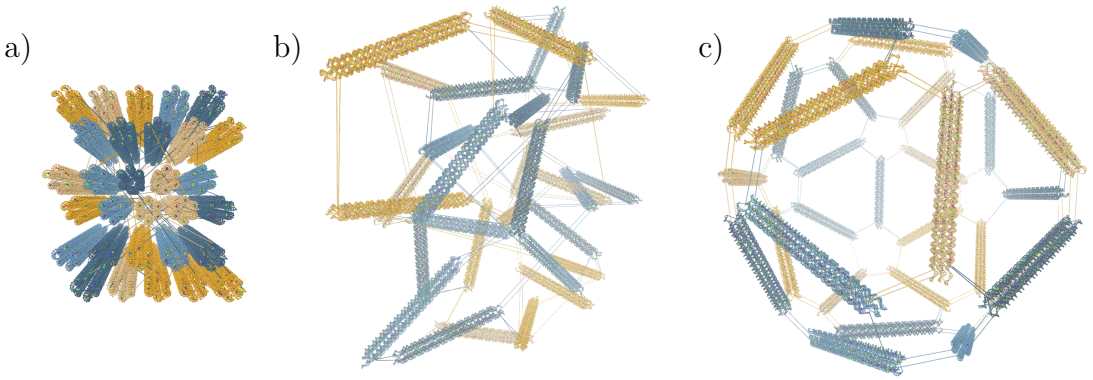

Figure 6.2: Rigid-body dynamics of clusters. Snapshots from the automatic rigid-body relaxation of an icosahedron, starting with the configuration converted from caDNAno **a)**, through the intermediate **b)** where the dynamics are applied, and **c)** the final resulting relaxed state.

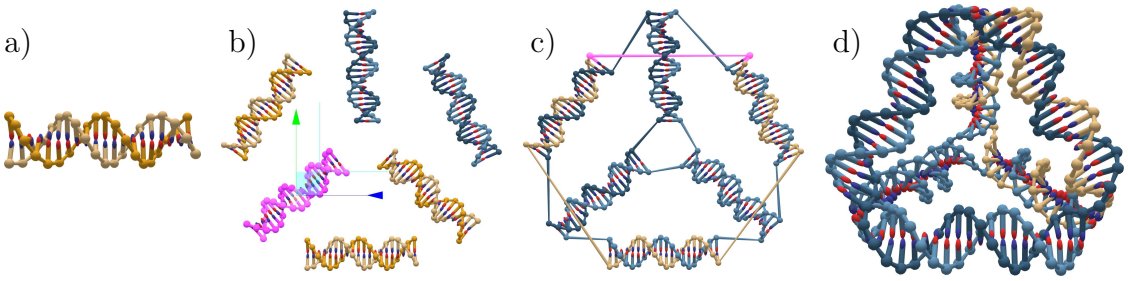

Figure 6.3: Designing the DNA tetrahedron from [68] using the oxView editing tools. **a)** An initial 20 base pair helix created. **b)** Duplicated helices being rotated and translated into place. **c)** Strands ligated together. **d)** The resulting 3D tetrahedron shape, as seen after applying rigid-body dynamics.

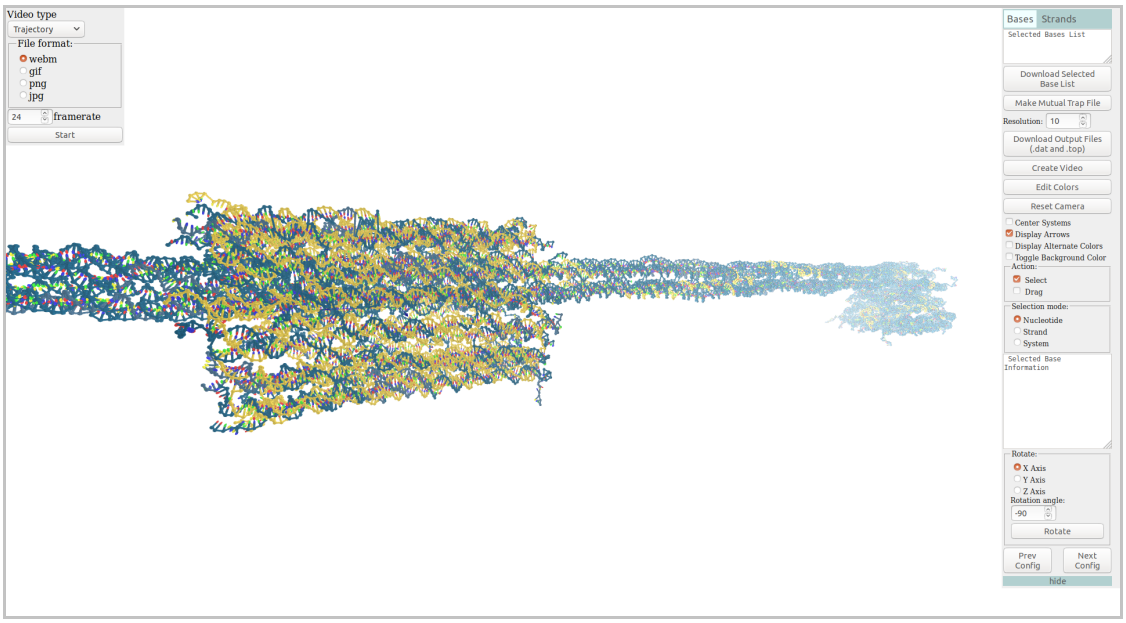

Figure 6.4: Screenshot of the online oxView tool, exporting a video of a slider on a rail from a simulated trajectory.

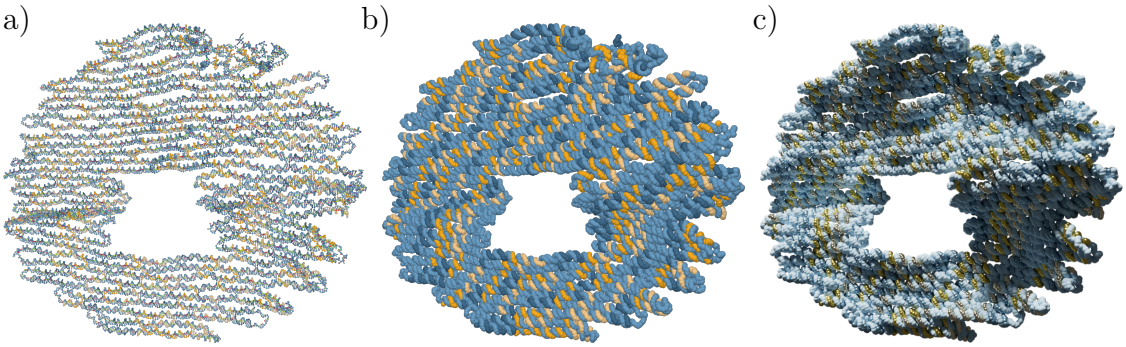

Figure 6.5: Component scaling and image export. **a)** Default oxView visualisation. **b)** Custom component scale and visibility. Using the"Visible components" dropdown in the "View" menu, the backbone spheres have been scaled up by a factor of 4.18 while all other nucleotide components have been hidden. **c)** The scaled scene in **b)** exported as glTF and rendered using Cycles in Blender.

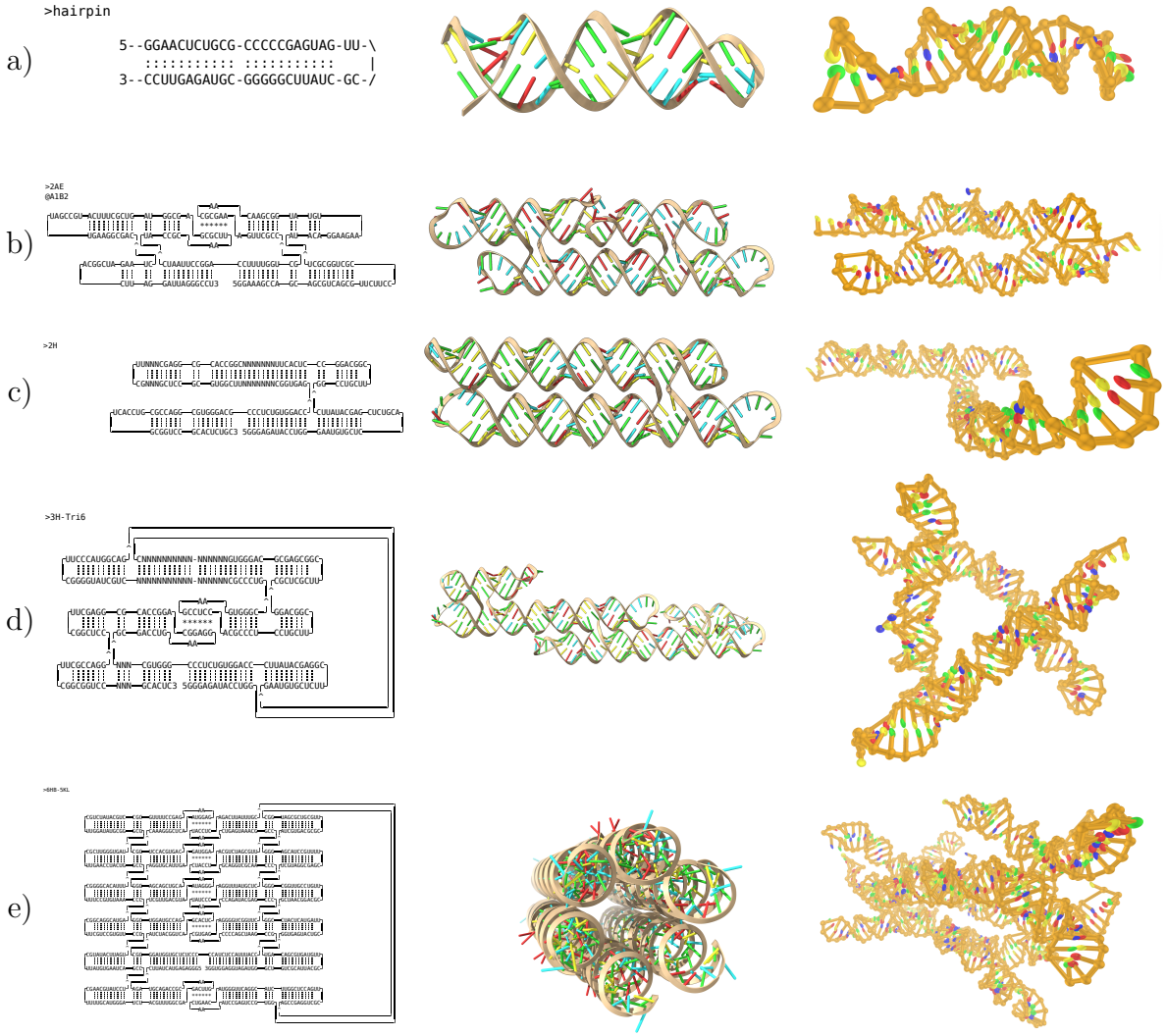

Figure 6.6: Conversion and simulation of various RNA designs. Each row, from left to right, shows the ASCII blueprint design, the PDB model (visualised using ChimeraX), and a frame from the simulated structure (visualised using oxView). **a)** Is a simple hairpin loop. **b)** is a two-helix bundle tile used in [7]. **c)** is two helices connected by a double crossover, analysing the flexibility of such a motif. **d)** is a possible design for a tensegrity triangle. **e)** is a siz-helix bundle.

You live and learn. At any rate, you live.

— Douglas Adams, Mostly Harmless

7 Conclusion

In conclusion, nanostructures self-assembly design has been investigated on both an abstract and a more detailed level. The presented projects have resulted in tools and methods for creating, simulating and analysing self-limiting modular structures with minimal complexity, potentially containing building blocks created in different design software.

7.1 Abstract self-assembly and design

The polycube model provides a simple way to self-assemble and design modular shapes. The stochastic assembler presented in Chapter 3 provides a method to quickly evaluate input candidates, ensuring that they give the intended output.

The same chapter also showed how large samples of the input space give valuable insight into the frequency of complex and simple shapes. With low complexity polycubes exponentially more common than their high complexity counterparts (during random sampling), we can imagine how simplicity in nature is not necessarily an evolutionary adaption but simply a more probable outcome.

While Chapter 3 showed how to map input rules into output shapes, Chapter 4 showed how to find the input rules that make a given shape, making it possible to

determine the minimal complexity rule for an intended design. This should facilitate the design of modular experimental structures such as those presented in Chapter 2.

7.2 Nucleic acid design and simulation

The oxView online tool presented in Chapter 6 has significantly simplified the setup, visualisation, and analysis of oxDNA simulations. The ability to import and combine structures from different tools facilitates the design of modular multi-component structures such as those presented in Chapter 5.

7.3 Future work

Other shapes than cubes.

Staged assembly

Given a maximum complexity - for example, a limited number of possible colours - what is the largest bounded structure that can deterministically assemble?

Rule out alternative solutions given by non-deterministic assembly via SAT

Appendices

A

The polycube codebase

Contents

The code used for polycube assembly can be found at `https://github.com/Akodiat/polycubes`.

A.1 Stochastic assembly code

A.1.1 C++

A.1.2 Python binding

A.1.3 JavaScript

A.2 Polycube solver

A.2.1 Python

In order to speedup sampling, the c++ binary can be called multiple times in parallell and merged with the merge python script. For the results covered in Chapter 3, the sampling was done on 100 concurrent nodes and merged.

The largest sampling done, the polyomino reference with 1e9 samples (described in Section 3.2.1), took a total of 72 hours, 57 minutes and 58 seconds to sample with 100 copies each performing 1e7 samples in parallel. The merging script then took another 197 hours, 40 minutes and 24 seconds to run, but it clearly saved a significant amount of time compared to running a single sampling for 100 times longer (which would take almost 7300 hours, or 304 days).

Finally the analysis script took another 301 hours, 44 minutes and 17 seconds to run.

took 359 hours, 9 minutes and 44 seconds

A.2.2 JavaScript

A.2.3 Patchy particle code

The modified version of oxDNA used to simulate torsional patchy particles can be found at `https://github.com/Akodiat/oxDNA_torsion`. To download and combile, run:

```
git clone https://github.com/Akodiat/oxDNA_torsion.git
cd oxDNA
mkdir build
cd build
cmake ..
```

```
make -j4
make romano
```

Simulation files for a given polycube shape can then be generated through the web console at https://akodiat.github.io/polycubes, by running:

```
getPatchySimFiles(
  '0700000705008687000000000', // Rule string
  1, // Number of assemblies
  'cube', // Name
  '/users/joakim/repo/oxDNA_torsion', // Path to oxDNA directory
  [.01,.02,.03,.04,.05,.06,.07,.08,.09,.1], // Temperature range
  0.1 // Density (used to generate configuration when the number
  // of assemblies is more than one)
)
```

A.3 Analysis

B

The oxView codebase

The code for oxView can be found at `https://github.com/sulcgroup/oxdna-viewer`. It is written in TypeScript and compiled into JavaScript. The 3D visualisation is done with the help of the Three.js library, while the graphical user interface uses the Metro 4.

To compile, you need both typescript and Node.js installed. Typing `npm install` should install all required node modules.

Type `tsc` to compile.

References

[1] Chris R Calladine and Horace Drew. *Understanding DNA: the molecule and how it works*. Academic press, 1997.

[2] Nadrian C. Seeman. *Structural DNA Nanotechnology*. Cambridge University Press, 2016.

[3] Nadrian C Seeman. "Nucleic Acid Junctions and Lattices". In: *Journal of Theoretical Biology* 99 (1982), pp. 237–247.

[4] Paul WK Rothemund. "Folding DNA to create nanoscale shapes and patterns". In: *Nature* 440.7082 (2006), p. 297.

[5] Thomas E Ouldridge, Ard A Louis, and Jonathan PK Doye. "DNA nanotweezers studied with a coarse-grained model of DNA". In: *Physical review letters* 104.17 (2010), p. 178101.

[6] Peixuan Guo. "The emerging field of RNA nanotechnology". In: *Nature nanotechnology* 5.12 (2010), p. 833.

[7] Cody Geary, Paul WK Rothemund, and Ebbe S Andersen. "A single-stranded architecture for cotranscriptional folding of RNA nanostructures". In: *Science* 345.6198 (2014), pp. 799–804.

[8] Steffen L Sparvath, Cody W Geary, and Ebbe S Andersen. "Computer-aided design of RNA origami structures". In: *3D DNA Nanostructure*. Springer, 2017, pp. 51–80.

[9] Cody Geary et al. "RNA origami design tools enable cotranscriptional folding of kilobase-sized nanoscaffolds". In: *Nature chemistry* 13.6 (2021), pp. 549–558.

[10] Ming Li and P. M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications [electronic resource]*. eng. Fourth edition. Texts in computer science. Cham, 2019.

[11] Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. "Input–output maps are strongly biased towards simple outputs". In: *Nature communications* 9.1 (2018), p. 761.

[12] Nadrian C Seeman. "Nucleic acid junctions and lattices". In: *Journal of theoretical biology* 99.2 (1982), pp. 237–247.

[13] Erik Winfree. "Algorithmic self-assembly of DNA". PhD thesis. California Institute of Technology, 1998.

[14] Erik Winfree et al. "Design and self-assembly of two-dimensional DNA crystals". In: *Nature* 394.6693 (1998), p. 539.

[15] Luvena L Ong et al. "Programmable self-assembly of three-dimensional nanostructures from 10,000 unique components". In: *Nature* 552.7683 (2017), pp. 72–77.

[16] Grigory Tikhomirov, Philip Petersen, and Lulu Qian. “Fractal assembly of micrometre-scale DNA origami arrays with arbitrary patterns”. In: *Nature* 552.7683 (2017), p. 67.

[17] Grigory Tikhomirov, Philip Petersen, and Lulu Qian. “Programmable disorder in random DNA tilings”. In: *Nature nanotechnology* 12.3 (2017), pp. 251–259.

[18] Klaus F Wagenbauer, Christian Sigl, and Hendrik Dietz. “Gigadalton-scale shape-programmable DNA assemblies”. In: *Nature* 552.7683 (2017), p. 78.

[19] Sungwook Woo and Paul WK Rothemund. “Programmable molecular recognition based on the geometry of DNA nanostructures”. In: *Nature chemistry* 3.8 (2011), pp. 620–627.

[20] Christian Sigl et al. “Programmable icosahedral shell system for virus trapping”. In: *Nature Materials* 20.9 (2021), pp. 1281–1289.

[21] Zhiwei Lin et al. “Engineering Organization of DNA Nano-Chambers through Dimensionally Controlled and Multi-Sequence Encoded Differentiated Bonds”. In: *Journal of the American Chemical Society* 142.41 (2020). PMID: 32902966, pp. 17531–17542. eprint: https://doi.org/10.1021/jacs.0c07263. URL: https://doi.org/10.1021/jacs.0c07263.

[22] Mingyang Wang et al. “Programmable Assembly of Nano-architectures through Designing Anisotropic DNA Origami Patches”. In: *Angewandte Chemie International Edition* 59.16 (2020), pp. 6389–6396. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.201913958.

[23] Hao Wang. “Proving theorems by pattern recognition—II”. In: *Bell system technical journal* 40.1 (1961), pp. 1–41.

[24] David Doty. “Theory of algorithmic self-assembly”. In: *Communications of the ACM* 55.12 (2012), pp. 78–88.

[25] David Doty. “DNA tile self-assembly”. Tutorial presented at the 23rd International Conference on DNA Computing and Molecular Programming. Sept. 2017. URL: https://web.cs.ucdavis.edu/~doty/papers/dna23-tile-assembly-tutorial.pdf.

[26] SE Ahnert et al. “Self-assembly, modularity, and physical complexity”. In: *Physical Review E* 82.2 (2010), p. 026117.

[27] Iain G Johnston et al. “Evolutionary dynamics in a simple model of self-assembly”. In: *Physical Review E* 83.6 (2011), p. 066105.

[28] Iain G Johnston et al. “Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution”. In: *bioRxiv* (2021). URL: https://www.biorxiv.org/content/early/2021/07/29/2021.07.28.454038.

[29] Lorenzo Rovigatti et al. “A comparison between parallelization approaches in molecular dynamics simulations on GPUs”. In: *Journal of computational chemistry* 36.1 (2015), pp. 1–8.

[30] Flavio Romano et al. “Designing patchy interactions to self-assemble arbitrary structures”. In: *Physical Review Letters* 125.11 (2020), p. 118003.

[31] A. Lempel and J. Ziv. “On the Complexity of Finite Sequences”. In: *IEEE Transactions on Information Theory* 22.1 (1976), pp. 75–81.

[32] NJA Sloane and Simon Plouffe. *The encyclopedia of integer sequences*. 1995.

[33] NJA Sloane and D. Hugh Redelmeier. *A000988 - Number of one-sided polyominoes with n cells*. URL: https://oeis.org/A000988 (visited on 04/24/2020).

[34] Kamaludin Dingle, Guillermo Valle Pérez, and Ard A Louis. "Generic predictions of output probability based on complexities of inputs and outputs". In: *Scientific reports* 10.1 (2020), pp. 1–9.

[35] Shawn M. Douglas et al. "Rapid prototyping of 3D DNA-origami shapes with caDNAno". In: *Nucleic Acids Research* 37.15 (2009), pp. 5001–5006.

[36] David Doty, Benjamin L Lee, and Tristan Stérin. "scadnano: A browser-based, scriptable tool for designing DNA nanostructures". In: *DNA 2020: Proceedings of the 26th International Meeting on DNA Computing and Molecular Programming*. Ed. by Cody Geary and Matthew J. Patitz. Vol. 174. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 9:1–9:17. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12962.

[37] Nick Conway and Shawn Douglas. *Windows Installation - caDNAno*. URL: https://cadnano.org/windows-installation.html (visited on 08/20/2021).

[38] Erik Benson et al. "DNA rendering of polyhedral meshes at the nanoscale". In: *Nature* 523.7561 (2015), pp. 441–444.

[39] Erik Benson et al. "Computer-Aided Production of Scaffolded DNA Nanostructures from Flat Sheet Meshes". In: *Angewandte Chemie International Edition* 55.31 (2016), pp. 8869–8872.

[40] Erik Benson. *vHelix - Free-form DNA-nanostructure design*. 2015. URL: http://www.vhelix.net/ (visited on 11/03/2021).

[41] Hyungmin Jun et al. "Rapid prototyping of arbitrary 2D and 3D wireframe DNA origami". In: *Nucleic Acids Research* (Sept. 2021). gkab762. eprint: https://academic.oup.com/nar/advance-article-pdf/doi/10.1093/nar/gkab762/40347509/gkab762.pdf. URL: https://doi.org/10.1093/nar/gkab762.

[42] Sean Williams et al. "Tiamat: A Three-Dimensional Editing Tool for Complex DNA Structures". In: *DNA Computing*. Ed. by Ashish Goel, Friedrich C Simmel, and Petr Sosík. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 90–101.

[43] Hao Yan. *Tiamat download*. 2008. URL: http://yanlab.asu.edu/Resources.html (visited on 11/03/2021).

[44] Haichao Miao et al. "Multiscale Visualization and Scale-adaptive Modification of DNA Nanostructures". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018). URL: https://www.cg.tuwien.ac.at/research/publications/2018/miao_tvcg_2018/.

[45] Chao-Min Huang et al. "Integrated computer-aided engineering and design for DNA assemblies". In: *Nature Materials* (2021), pp. 1–8.

[46] James C Phillips et al. "Scalable molecular dynamics with NAMD". In: *Journal of computational chemistry* 26.16 (2005), pp. 1781–1802.

[47] Wendy D Cornell et al. “A second generation force field for the simulation of proteins, nucleic acids, and organic molecules J. Am. Chem. Soc. 1995, 117, 5179-5197”. In: *Journal of the American Chemical Society* 118.9 (1996), pp. 2309–2309.

[48] Bernard R Brooks et al. “CHARMM: a program for macromolecular energy, minimization, and dynamics calculations”. In: *Journal of computational chemistry* 4.2 (1983), pp. 187–217.

[49] Jejoong Yoo and Aleksei Aksimentiev. “In situ structure and dynamics of DNA origami determined through molecular dynamics simulations”. In: *Proceedings of the National Academy of Sciences* 110.50 (2013), pp. 20099–20104.

[50] Aditya Sengar et al. “A primer on the oxDNA model of DNA: When to use it, how to simulate it and how to interpret the results”. In: *arXiv preprint arXiv:2104.11567* (2021).

[51] Rahul Sharma et al. “Characterizing the motion of jointed DNA nanostructures using a coarse-grained model”. In: *ACS nano* 11.12 (2017), pp. 12426–12435.

[52] Petr Šulc et al. “A nucleotide-level coarse-grained model of RNA”. In: *The Journal of chemical physics* 140.23 (2014), 06B614_1.

[53] Thomas Gerling et al. “Dynamic DNA devices and assemblies formed by shape-complementary, non–base pairing 3D components”. In: *Science* 347.6229 (2015), pp. 1446–1452.

[54] Reza M Zadegan et al. “Construction of a 4 zeptoliters switchable 3D DNA box origami”. In: *ACS nano* 6.11 (2012), pp. 10050–10053.

[55] Tim Liedl et al. “Self-assembly of three-dimensional prestressed tensegrity structures from DNA”. In: *Nature nanotechnology* 5.7 (2010), p. 520.

[56] Dongran Han et al. “Folding and cutting DNA into reconfigurable topological nanostructures”. In: *Nature nanotechnology* 5.10 (2010), p. 712.

[57] Carlos Ernesto Castro et al. “A primer to scaffolded DNA origami”. In: *Nature methods* 8.3 (2011), p. 221.

[58] Do-Nyun Kim et al. “Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures”. In: *Nucleic acids research* 40.7 (2012), pp. 2862–2868.

[59] Antonio Suma et al. “TacoxDNA: A user-friendly web server for simulations of complex DNA structures, from single strands to origami”. In: *Journal of Computational Chemistry* (2019). URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.26029.

[60] Erik Poppleton et al. “Design, optimization and analysis of large DNA and RNA nanostructures through interactive visualization, editing and molecular simulation”. In: *Nucleic acids research* 48.12 (2020), e72–e72.

[61] Antonio Suma et al. “TacoxDNA: A user-friendly web server for simulations of complex DNA structures, from single strands to origami”. In: *Journal of computational chemistry* 40.29 (2019), pp. 2586–2595.

[62] Erik Benson et al. “Strategies for Constructing and Operating DNA Origami Linear Actuators”. In: *Small* 17.20 (2021), p. 2007704.

[63] Jonah Procyk, Erik Poppleton, and Petr Šulc. “Coarse-grained nucleic acid–protein model for hybrid nanotechnology”. In: *Soft Matter* 17.13 (2021), pp. 3586–3593.

[64] Chris Maffeo and Aleksei Aksimentiev. *Multi-resolution simulations of self-assembled DNA nanostructures*. 2018. URL: https://gitlab.engr.illinois.edu/tbgl/tutorials/multi-resolution-dna-nanotechnology/tree/master.

[65] Chao-Min Huang et al. "Uncertainty quantification of a DNA origami mechanism using a coarse-grained model and kinematic variance analysis". In: *Nanoscale* 11.4 (2019), pp. 1647–1660.

[66] David Baraff. *An introduction to physically based modeling: Rigid Body Simulation I — Unconstrained Rigid Body Dynamics*. 1997.

[67] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Kdd*. Vol. 96. 34. 1996, pp. 226–231.

[68] Russell P Goodman et al. "Rapid chiral assembly of rigid DNA building blocks for molecular nanofabrication". In: *Science* 310.5754 (2005), pp. 1661–1665.

[69] Linge Bai and David Breen. "Calculating Center of Mass in an Unbounded 2D Environment". In: *Journal of Graphics Tools* 13.4 (2008), pp. 53–60. URL: https://doi.org/10.1080/2151237X.2008.10129266.