Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Antti Elonen

Algorithmic design of RNA polyhedra

Master's Thesis
Espoo, September 12, 2018

Supervisor: Professor Pekka Orponen
Advisor: Abdulmelik Mohammed M. Sc.(Tech.)

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Antti Elonen		
Title:	Algorithmic design of RNA polyhedra		
Date:	September 12, 2018	**Pages:**	56
Major:	Machine Learning and Data Mining	**Code:**	SCI3044
Supervisor:	Professor Pekka Orponen		
Advisor:	Abdulmelik Mohammed M. Sc.(Tech.)		

The field of bottom-up nanotechnology has been the subject of much research in the recent years. Most of that research has focused on creating nano-scale shapes and structures using multiple strands. DNA origamis and various tile-based schemes are perhaps the most famous examples. No such robust design schemes exist for the design of single stranded RNA structures, however, despite their potential to offer a cheap and sound approach to nanomanufacturing.

In this thesis, we study the problem of designing single-stranded RNA polyhedral wireframes, i.e., such RNA strands that fold into the wireframe of a given polyhedron. We introduce a kissing-loop based design scheme, which routes an RNA strand around a spanning tree of a polyhedron, and we show how to do the routing on arbitrary polyhedra while avoiding knots. We also introduce a design tool, *Sterna*, which is based on these principles. It allows the user to convert a 3D model of a polyhedron into an RNA secondary and tertiary structures, which can be further developed into a primary structure with the additional scripts we have provided. Finally, we design three RNA polyhedra, which are synthesized and imaged in a project related to this master's thesis. The resulting images lend credence to the soundness of Sterna and the underlying design process.

Keywords:	RNA, Nanotechnology, Design, Polyhedra, Wireframe, Spanning tree, Sterna
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Antti Elonen		
Työn nimi:	RNA polyhedrojen algoritminen suunnittelu		
Päiväys:	12. syyskuuta 2018	**Sivumäärä:**	56
Pääaine:	Machine Learning and Data Mining	**Koodi:**	SCI3044
Valvoja:	Professori Pekka Orponen		
Ohjaaja:	DI Abdulmelik Mohammed		

Yksi koostavan (engl. bottom-up) nanoteknologian keskeisiä tutkimusalueita viime vuosina on ollut DNA-nanoteknologia, so. nanokokoisten kappaleiden ja rakennelmien tuottaminen biopolymeereistä. Niinsanotut DNA-origamit ja -laatoitukset ovat tämän lähestymistavan tunnetuimpia esimerkkejä. Vastaavaa yleistä menetelmää ei toistaiseksi ole ollut nanorakenteiden tuottamiseen yksisäikeisistä RNA-polymeereistä, vaikka nämä periaatteessa tarjoaisivat edullisen ja skaalautuvan lähtökohdan nanovalmistukselle.

Tässä diplomityössä tarkastelemme 3D-monitahokkaiden rautalankamallien laskostamista yksisäikeisistä RNA-polymeereistä. Kehitämme automatisoidun suunnitteluprosessin, joka tuottaa syötteenä annettua monitahokasta vastaavaan muotoon laskostuvan RNA-emästen jonon. Käyttämämme menetelmä perustuu RNA-säikeen reitittämiseen monitahokkaan virittävän puun ympäri ja rakenteen sulkemiseen ns. silmukkapareilla (engl. kissing loop motif). Esitämme myös, miten mielivaltaisen monitahokkaan virittävä puu on mahdollista reitittää tuottamatta topologisia solmuja, jotka estäisivät vastaavan RNA-polymeerin laskostumisen.

Toteuttamamme *Sterna*-suunnitteluohjelman avulla käyttäjä voi tuottaa mistä tahansa 3D-monitahokasmallista sen muotoon laskostuvan RNA-jonon sekundääri- ja tertiäärirakennekuvaukset. Tarjoamme myös ohjelman jonka avulla nämä voidaan edelleen täydentää emästiedoilla biosynteesiä varten tarvittavaksi RNA-primäärirakenteeksi. Käyttöesimerkkeinä suunnittelemme kolme RNA-monitahokasta, jotka on syntetisoitu ja kuvannettu tämän diplomityön kumppanihankkeissa. Saadut tulokset todentavat suunnittelumenetelmämme ja siihen pohjautuvan Sterna-työkalun oikeellisuutta.

Asiasanat:	RNA, Nanoteknologia, Suunnittelu, Monitahokkaat, Rautalankamalli, Virittävä puu, Sterna
Kieli:	Englanti

Acknowledgements

I wish to thank my supervisor Pekka Orponen for giving me this opportunity, for guiding me through it and for his exacting review of this thesis. I also want to thank Abdulmelik Mohammed for his guidance and help. Lastly, I give my thanks to everyone else involved in this project for their good work and will to see it finished.

Espoo, September 12, 2018

Antti Elonen

Contents

Chapter 1

Introduction

The DNA double helix might be the best known and most iconic symbol of nanoscience. It consists of two DNA strands circling around each other, each base of one connected to those of the other at equal intervals in complexes known as base pairs. It is perhaps less known that these two strands might actually be one. In a process called molecular folding, biopolymers wrap around themselves to create intricate shapes and structures at nanometer scale. This process can turn even just a single strand into a truly complex three dimensional structure [20, p. 26].

The approach of creating nanostructures through the self assembly or synthesis of nanoscale building blocks is called the *bottom-up* approach. This is the way all life is formed in nature. Bottom-up approach is the opposite of top-down approach, which attempts to reach the same goals by shrinking larger structures through etching, cutting or sculpting. The main benefits of bottom-up approach are potentially lower costs, higher scalability and efficient parallel processing [14, pp. 66–106].

One of the more famous approaches to bottom-up nanotechnology is Paul Rothemund's DNA-origami [16], which forces a long strand of DNA to desired conformations by attaching to it numerous short DNA-strands called staple strands. This approach has been very successful, but it requires the synthesis of a great number of different strands. In 2016, Veneziano et al. published spanning tree based designs, which greatly eased the design process for DNA-origamis [22].

DNA-origamis, tiles and other methods have been used to create a number of exceptionally complex nanoscale structures, including Platonic solids, maps and even tiny bunnies and nanoscale computers. Despite these advancements in DNA nanotechnology, RNA still lacks behind. While the most complex structures created in DNA consist of hundreds of thousands of nucleotides, their RNA counterparts are still constrained to a fraction of

that [10, 19].

RNA might nevertheless be a more suitable material for nanotechnology than DNA due to its greater structural variety and the possibility of synthesizing it through cloning.[8] This thesis focuses on single-stranded RNA structures that do not use any staple strands. With only one strand to synthesize, this could eventually be a simpler process than the synthesis of DNA-origamis, which may require hundreds of different strands. Presently, however, it actually is easier to synthesize numerous short strands rather than a single longer one. Nevertheless, single-stranded RNA can theoretically be synthesized even at room temperature by cellular processes and continuously cloned inside the nucleus of a cell, which would not be possible for multiple strands.

1.1 Problem statement

In this thesis, we try to solve the problem of algorithmically designing single RNA strands that fold into wireframes with the topology of a target polyhedron. This involves modelling the RNA strand, routing it around the polyhedron and designing the final sequence. Our ultimate goal is to create an automated process of converting a 3D-model into a sequence of nucleotides that will fold into a wireframe of the given model when synthesized, as depicted in Figure 1.1.

This thesis will introduce new algorithms and tools for the design of three dimensional structures using a single strand of RNA. We present a spanning tree based routing algorithm to guide a strand of RNA into taking the shape of almost any arbitrary polyhedron, and we provide new design tools, Sterna and snacseq, to convert almost any given 3D-model into an RNA-strand, which should fold into the wireframe of the given model.

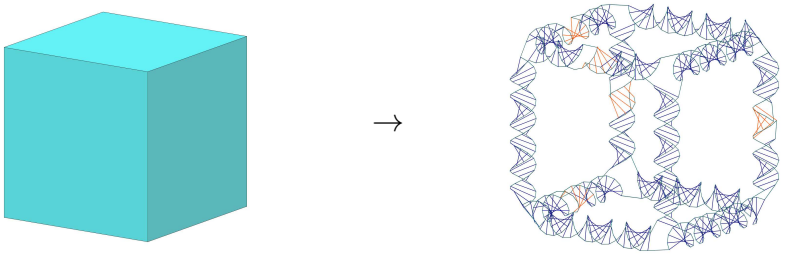


Figure 1.1: The essence of the thesis distilled into one image: Converting a 3D-model to an RNA strand folding into its wireframe.

1.2 Structure of the thesis

This thesis is divided into seven chapters. In the first two, we introduce the problem of designing RNA polyhedra and lay the foundations for our work. We give a brief overview of the field and explain key concepts relevant to RNA from a computer science point of view.

In Chapter 3, we focus on finding single stranded RNA secondary structures that correspond to arbitrary polyhedra. In Section 3.2, we show how to route a strand around any spanning tree without forming topological knots. We introduce a sorting algorithm called zig-zag combing in Section 3.2 and use it in conjunction with depth first search to find an unknotted path around a polyhedron. Section 3.3 explains how we generate an RNA strand along this path based on the P-stick model by Geary and Andersen [9]. We also show how to use a spring relaxation algorithm to orientate double helices along the edges of the polyhedra.

Chapter 4 describes how we convert secondary structures into final RNA sequences by splitting the primary structure into a stem and kissing loops and by generating them separately with NUPACK [26]. Section 4.1 introduces constraints on the stem exerted by DNA synthesis and transcription. In Section 4.2, we discuss kissing loops, their orthogonality and their generation. We also present a greedy algorithm for generating sets of orthogonal kissing loops.

Chapter 5 introduces Sterna, a secondary structure design tool based on the principles, procedures and algorithms described in the previous chapters. In Sections 5.1, 5.2 and 5.3, we explain the capabilities and input and output formats of Sterna and introduce the snac file format. Section 5.4 presents three additional scripts to generate pseudoknots and primary structures and to convert RNA strands from snac format to oxDNA input files for further analysis.

Finally, in the results chapter (Chapter 6), we discuss how we designed and simulated three single stranded RNA polyhedra, which were synthesized by our collaborators Ibuki Kawamata and Lukas Oesinghaus and imaged by Ibuki Kawamata. Section 6.2 presents the laboratory results and electron microscope images of the polyhedra. The final chapter, Chapter 7, concludes the thesis, sums up our work and suggests interesting problems for future studies.

Chapter 2

Background

2.1 RNA

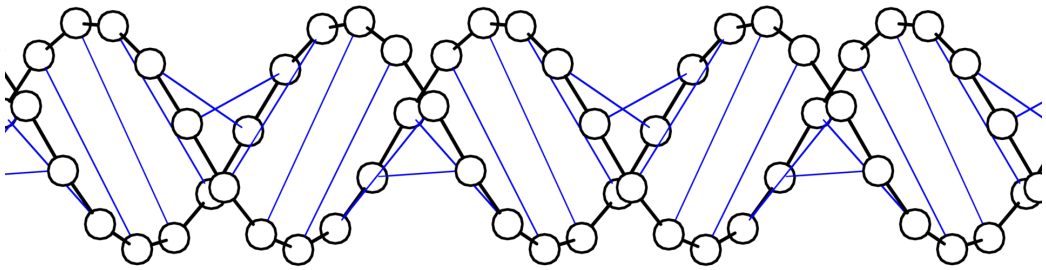


Figure 2.1: A P-stick model of an RNA helix

RNA (ribonucleic acid) is one of the principal biopolymers essential to life, alongside DNA (deoxyribonucleic acid) and proteins. RNA and its more famous cousin, DNA, are both nucleic acids, i.e., they consist of a chain of nucleotides. In RNA, these nucleotides come in four flavors, the bases *adenine*, *uracil*, *guanine* and *cytosine*, denoted with letters A, U, G and C respectively. The order and the number of the nucleotides defines the *primary structure* of the molecule, and it determines how the strand folds and interacts with other molecules [18, pp. 20–22]. RNA is created in a process called *transcription*, where the enzyme RNA polymerase binds to a DNA template and creates the RNA-strand one nucleotide at a time [7, p. 11].

Since the primary structure is defined with the language of four bases, it can be easily represented as a simple string of A's, U's, G's and C's. This

string starts from the 5' (five-prime) end of the RNA strands and ends at the 3' (three-prime) end. This order is imposed by 3'-5' phosphodiester bonds joining riboses together to create the backbone of the RNA strand [20, p. 22]. Although four letters are enough to fully describe the primary structure of an RNA strand, it is often useful to specify the bases less rigidly: Nucleic acid notation expands this language by twelve additional characters.

Symbol	Description	Bases
A	Adenine	A
C	Cytosine	C
G	Guanine	G
U	Uracil	U
W	Weak	A, U
S	Strong	C, G
M	aMino	A, C
K	Keto	G, U
R	puRine	A, G
Y	pYrimidine	C, U
B	not A	C, G, U
D	not C	A, G, U
H	not G	A, C, U
V	not U	A, C, G
N	any Nucleotide	A, C, G, U
Z	Zero	None

Table 2.1: Nucleic Acid Notation Table

A strand of RNA folds around itself, since its constituent bases interact with each other to form base pairs with specific hydrogen bonds. The canonical base pairs, also called *Watson-Crick* base pairs, are guanine-cytosine, adenine-uracil and so-called *wobble-pair*, guanine-uracil[18, p. 23, p. 211]. In special circumstances, such as at the end of a helix or when flanked by no other base pairs, any two bases can form non-canonical pairs [17, pp. 15–16]. The strength of a base pair connection depends not only on the bases themselves but also on the surrounding bases and their orientations. Base pair strengths are commonly modeled with the *nearest neighbor model* of RNA energetics [25]. It assigns energy values for all base pairs according to a large experimentally derived rule set of short sequences pairing up with other sequences.

The base pairs of an RNA strand are most often represented using the *dot-bracket-notation* as shown in the caption of Figure 2.2. In dot-bracket-notation, dots represent unpaired bases, opening brackets represent the first base of a pair, and closing brackets represent the second base of a pair. The *secondary structure* of an RNA strand is defined as the base pairs that can be depicted with dot-bracket-notation and these three characters. More formally, if the primary structure is defined as an indexed array, the secondary structure P is defined as a set pairs $\langle i, j\rangle \in P, i < j$, where $\forall\langle p_1, p_2\rangle, \langle q_1, q_2\rangle \in P : p_1 < q_1 < p_2 \leftrightarrow q_2 < p_2$.

As shown in Figure 2.2 and Figure 2.3, base pairs can also be illustrated with an *arc diagram*. An arc diagram of an RNA strand is constructed by laying its primary structure on a single line and by drawing an arc for each base pair. All arc diagrams with no overlapping arcs can be equivalently

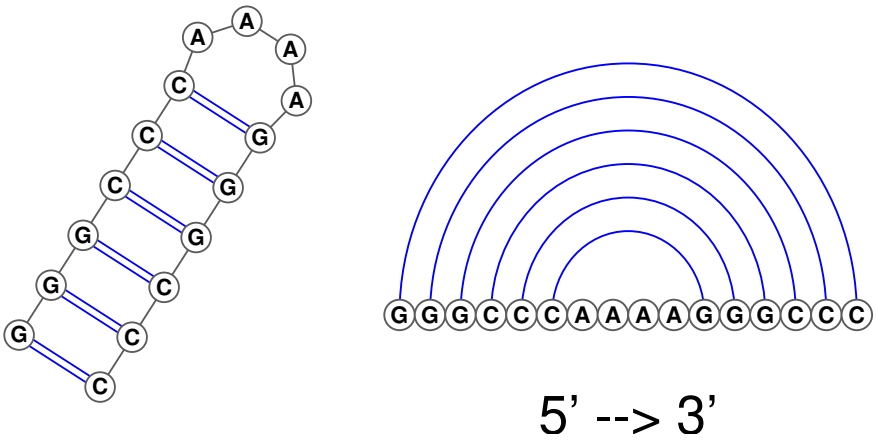


Figure 2.2: Left: A secondary-structure diagram of a loop with six basepairs and four unpaired bases. Right: An arc diagram of the same structure. The dotbracket-notation for this structure would be "((((((....))))))".

represented with dot-bracket-notation. *The tertiary structure* of an RNA strand is defined in relation to the secondary structure as the base pairs that do not fit the definition of the secondary structure. Equivalently, the tertiary structure consists of the unnested base pairs, or *pseudoknots*, i.e., as those pairs denotable only by crossing arcs in the arc diagram.

2.2 RNA models

As previously mentioned, the strength of base pairs can be modeled using the nearest neighbor model. The sum of the strengths of all base pairs defines the total free energy of the secondary structure. The second law of thermodynamics necessitates that the RNA strand tends towards the structure that minimizes its free energy. By finding this structure, we can predict the final conformation of the strand in nature. We can therefore in principle predict secondary structures based on primary structures or even design primary structures based on given secondary structures.

The nearest neighbor model is a simplified thermodynamic model based on observed energies of RNA strands. In principle, it defines a set of all possible pairs of two base pairs and assigns an energy value to them. The energy of the secondary structure can be calculated by summing up all of these pairs occurring in it [12, pp. 20–23]. The actual nearest neighbor model contains multiple additional rules and exceptions, but they are irrelevant for the understandability of this thesis.

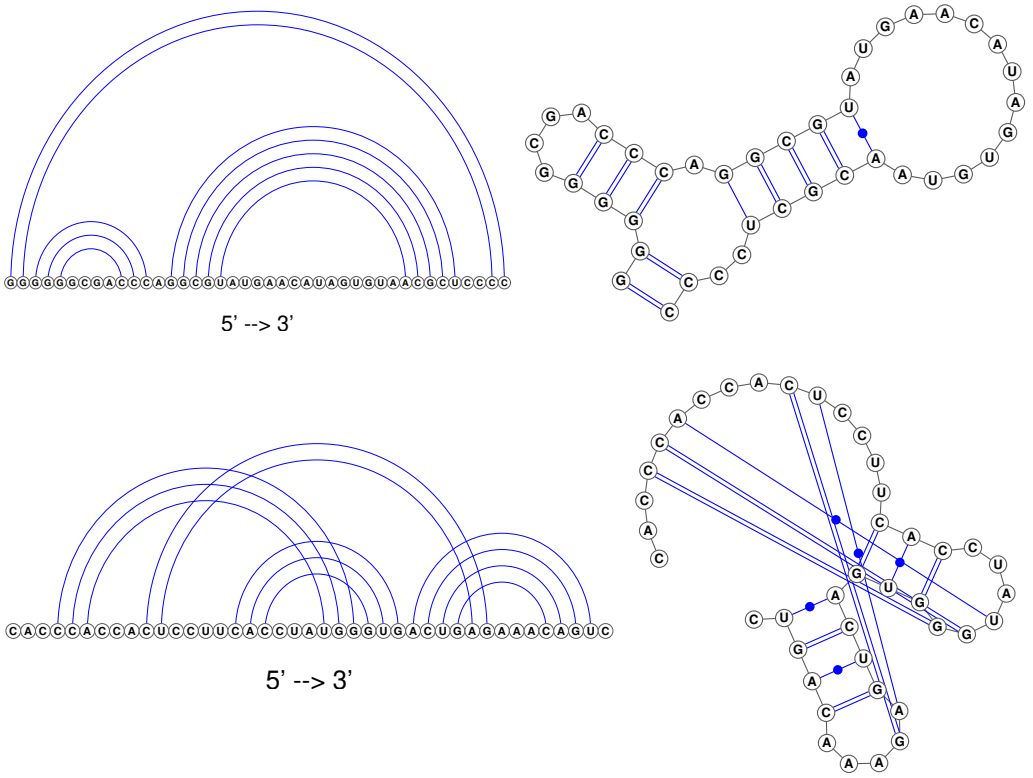


Figure 2.3: First row: A well nested RNA strand with loop and helix secondary structure motifs. Second row: An RNA strand with pseudoknots and a tertiary structure.

Parameter	Variable	A-form
Radius	R	8.7 Å
Rise	D	2.81 Å
Inclination	I	-7.45 Å
Axis	A	139.9 Å
Twist	T	32.73 Å
Helicity	H	11 bp

Table 2.2: P-stick model parameters. Adapted from the 2014 paper by Geary and Andersen [9].

The geometry of RNA is similar to that of DNA. It will also form helical structures, albeit with slight differences. Since atomic models of RNA would be unreasonably complicated and almost impossible to simulate with modern computers [12, 43–66], we abstract RNA by using the *P-stick helix model* as defined by Geary and Andersen [9]. In this model, an RNA strand is thought of as a series of beads, whose placement is determined by the Equation 2.1.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p \cdot D \\ R \cdot cos(p \cdot \frac{2\pi}{H}) \\ R \cdot sin(p \cdot \frac{2\pi}{H}) \end{bmatrix}, p \in \mathbb{Z} \tag{2.1}$$

The parameters for the P-stick model are listed in Table 2.2. Reality is not quite this well parameterized, however, as the helicity of RNA is not fixed. There can be significant variance in the exact form between bases even in the same helix. These parameters are therefore only averages of clusters of natural RNA molecules derived by Geary and Andersen. In this thesis, we consider all RNA helices to be of the A-form, which is the most common form in nature [20, pp. 23–25].

2.3 Building with RNA

An RNA secondary structure consists of different structural motifs. These are, for instance, double-helix, hairpin-loop, multi-loop, bulge and internal loop. Hairpin loops can be thought of as U-turns and multi-loops as junctions between helices. An internal loop is a special bulge, where both sides of a double helix are bulged with unpaired bases [17, pp. 31–34]. Three of these motifs are illustrated in Figure 2.4. All of the secondary structures in this thesis are combinations of loops, double helices and multiloops. However, bulges, internal loops and possibly some more exotic motifs can occur in misfolded structures.

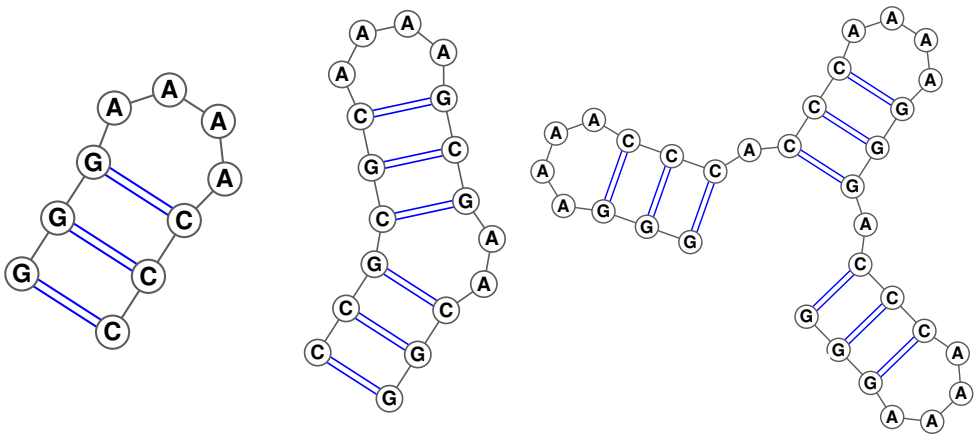


Figure 2.4: Two-dimensional representations of a stem loop, a stem loop with a bulge and a multi-loop. In dotbracket notation, these structures would be denoted with "(((....)))", "(((..(((....)))))))" and "(((....))).(((....))).(((....)))", respectively.

Pseudoknots form the tertiary structure of RNA. They consist of loops or bulges or any other unpaired bases in the secondary structure pairing up with each other. A kissing loop is a type of pseudoknot, where two hairpin

loops pair up and "kiss" each other. If the angle between these loops is 180 degrees, the kissing loop forms a continuous double helix as depicted in Figure 2.5.

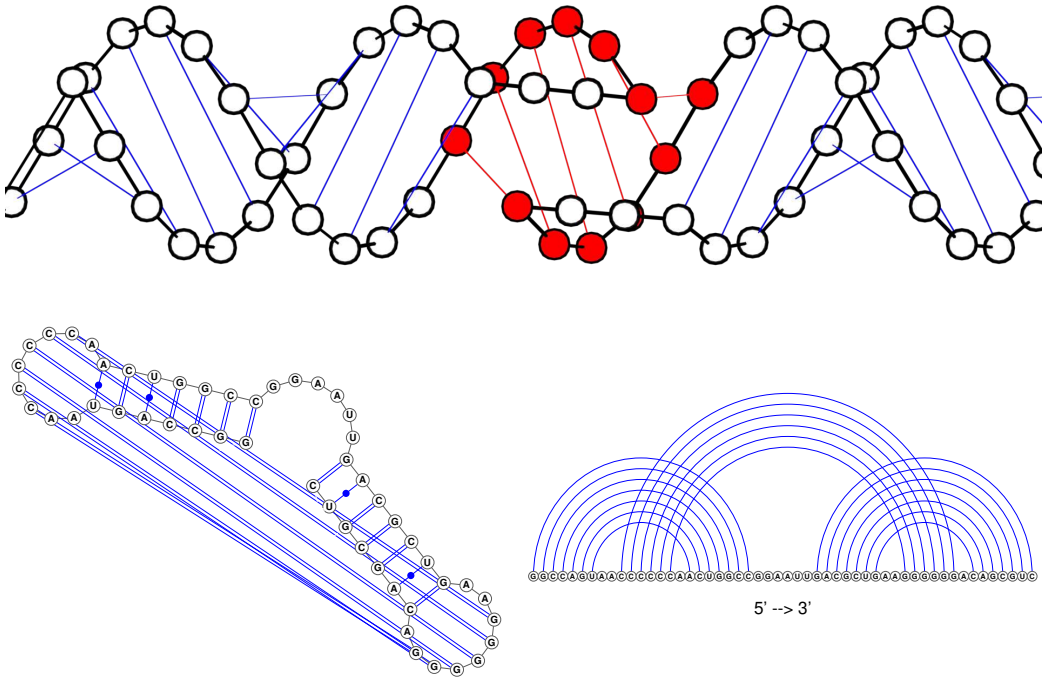


Figure 2.5: Kissing loop geometry and its 2D representations. The extended dotbracket notation for this structure is "(((((((..[[[[[[.)))))))).....(((((((..]]]]]].)))))))".

As previously hinted, pseudoknots and kissing loops cannot be represented with the traditional dotbracket-notation, since it would be impossible to deduce which open bracket a closing bracket corresponds to. Therefore, dotbracket-notation is often extended with square brackets to account for kissing loops. Furthermore, additional extensions are necessary for nonnested pseudoknots, i.e., in case some pseudoknot closes only after another pseudoknot begins. However, dotbracket-notation with square brackets is often enough, since pseudoknots are usually nested and rarely cross this way. Nested and nonnested strands are depicted in Figure 2.3. Only the first one of these can be represented with unextended dotbracket notation.

In this thesis, we are interested only in wireframe representations of polyhedra, that is, polyhedral beam networks, since filling the volume or even only surfaces of an entire shape would require exceedingly long RNA strands. To create an RNA wireframe of a polyhedron, we must route the RNA strand

around it in such a way that the structure stays rigid and that the *folding pathway* is feasible. The folding pathway of an RNA strand refers to the set of secondary and tertiary structures the strand transitions through on its way to the final structure. Some folding pathways might include such local energy optima that reaching the intended structure would be unlikely or almost impossible.

The stiffness of a polymer is characterized by its *persistence length* [1]. RNA double helices significantly shorter than their persistence length of about 60 nm can be thought of as rigid sticks and the junctions between them as almost freely articulated linkings. If we were to replace all the edges of a polyhedron with double helices, we could recreate it with RNA. However, these helices cannot be connected to one another in such a way that they form a single strand and recreate the intended topology, because at least one of the double helices would end in a loop, i.e., the double helix would not be fixed in place.

By replacing some edges of the polyhedron with helices and some with kissing loops, we actually can recreate a wireframe of the polyhedron with a single strand. Since a *spanning tree* (the smallest tree covering all vertices of a graph) of a polyhedron has no cycles, we can always route a strand around it. We can then fix some of the vertices together with kissing loops to recreate the non-spanning-tree edges. This way the edges in the spanning tree become helices and the non-spanning-tree edges become kissing loops. This procedure has been used in various papers on DNA nanostructure design, although usually without explicitly mentioning spanning trees [15].

Chapter 3

Design

In this chapter we introduce the process of converting a polyhedron into a viable RNA secondary structure. The process can be divided in three steps, modeling, path finding and generation.

3.1 Modeling

The first step of converting a polyhedron to an RNA strand is to choose the polyhedron. Our algorithm will work for all connected graphs lineraly embedded in three dimensional Euclidian space, but for simplicity we consider only three dimensional solids as valid input here. Even though we allow all 3D solids as input, some concave objects might lead to unrealistic results, like strands colliding or going through one another. Just a few of the possible polyhedra are depicted in Figure 3.1. The complexity of the object can span from anything as simple as a tetrahedron to a tesseract or even the head of a monkey.

It might be necessary to triangulate the polyhedron to make the final structure rigid, i.e., such that it cannot be deformed in any way, although triangulation is by itself not quite enough to guarantee rigidity [5, 6]. If rigidity is important, internal struts should be considered. The drawback of struts is that they complicate the structure and make it therefore harder to generate and less likely to fold properly.

3.2 Path finding

In the path finding step we choose a spanning tree of the polyhedron and trace a path twice around it to create a wireframe of the whole model via a single path (Figure 3.3). The actual bases will eventually be placed along

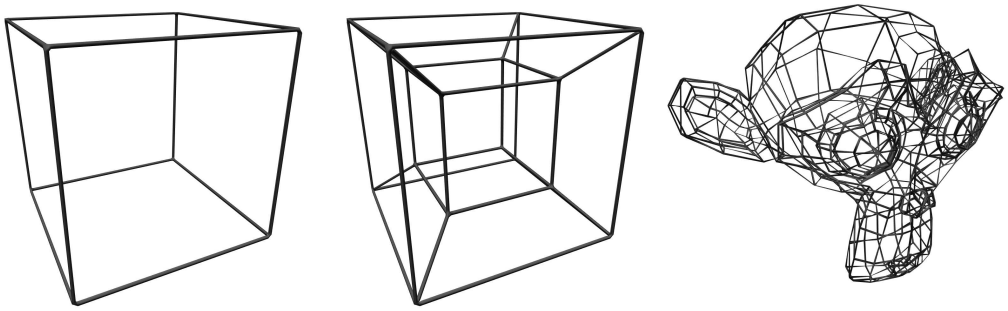


Figure 3.1: Three possible polyhedra.

this path, so it will dictate the ordering of the structural elements and the folding pathway of the final strand. Since the choice of the spanning tree will affect the folding pathway, some spanning trees might lead to more consistently correctly folding structures. Therefore, this choice is not completely inconsequential.

The path-tracing is done by the way of a depth first search, where the next vertex is always chosen in such a way as to avoid topological knots. The reason we want to avoid knots is to make the folding pathways feasible. Any knots will make the structure less likely to fold properly, since they force the folding strand to navigate through the knots, i.e., it would at best be like guiding a thread through the eye of a needle. Since electrically charged particles repel each other according to Coulomb's law, atoms repel each other at small distances, making it potentially impossible to fold knotted structures. In case of cotranscriptional folding, the choice of a spanning tree is especially important because of the sheer size of the RNA polymerase [13].

To avoid topological knots, we use the so called *zig-zag combing* algorithm. It sorts the adjacent edge at each vertex as we traverse the tree. The algorithm is exemplified in Figure 3.2 and explained step by step in the following paragraph. Python code for the zig-zag combing algorithm is given in Listing 3.1.

1. Create a sphere with radius of some small r at the vertex.

2. Cut all edges by r, i.e., by whatever of them falls within the sphere. This way, the starting points of all edges will fall onto the surface of a sphere, and they all have a certain latitude and longitude with regard to some axis.

3. Sort the edge incidence point into groups in an increasing order based on their longitude.

4. Sort the points in each group based on their latitude. Alternate between increasing and decreasing order to zig-zag along the sphere.

```
def zig_zag_comb(incoming_edge):
  latitudes = {}
  for e in vertex.edges:
      direction = e.verts[0].co - e.verts[1].co
      phi = math.atan2(direction.y, direction.x)
      theta = math.acos(direction.z)
      latitudes.setdefault(phi, []).append((e, theta))
  sorted_edges = []
  for i, phi in enumerate(sorted(latitudes.keys())):
    sorted_edges.extend([x[0] for x in sorted(latitudes[phi],
 key=itemgetter(2), reverse=(-1)**i)])
  return sorted_edges
```

Listing 3.1: Python code for the zig-zag combing algorithm:

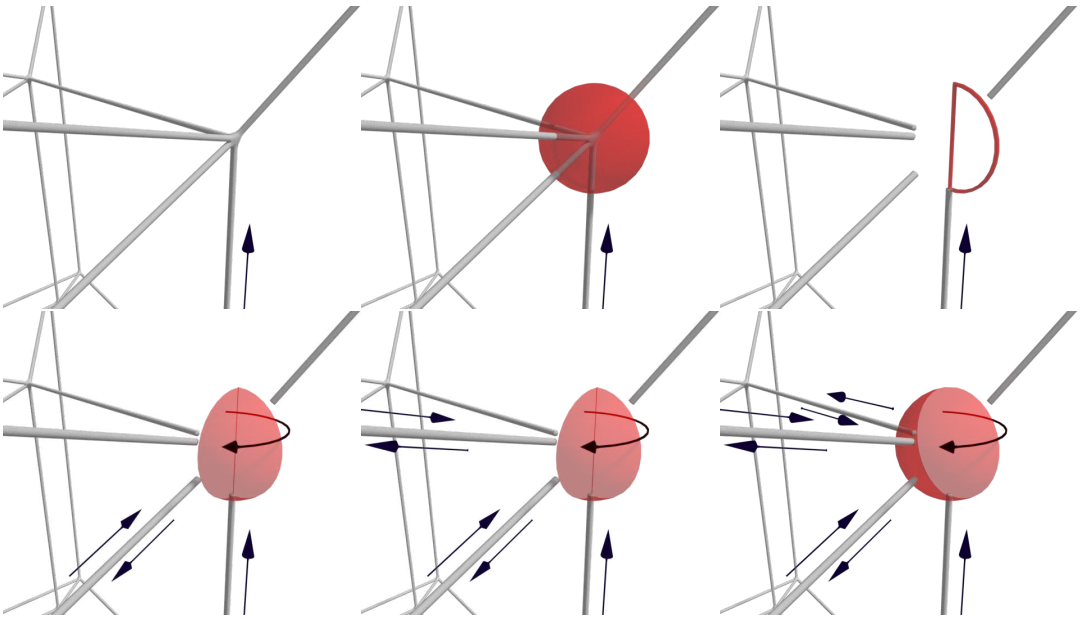


Figure 3.2: An example of the zig-zag combing algorithm.

This algorithm can be shown to result in an unknotted strand by considering how the strand is laid on the surface of the sphere. The strand will zig-zag across the sphere's surface from one pole to the other until it reaches the starting point again. This will naturally result in a non-crossing cycle. A sphere can be mapped to a plane by puncturing a hole in it and by stretching it across the plane, i.e., the non-crossing cycle on the sphere will be non-crossing also on a plane. Since a non-crossing cycle on a plane is

topologically equivalent to a circle, it is an unknot and therefore the strand itself must also be an unknot. Since the sum of unknots is an unknot and as the strand is an unknot at every vertex, the strand must be an unknot across all vertices and, therefore, across the whole structure.

The algorithm for routing a path twice around the spanning tree via depth first search, utilizing the zig-zag combing algorithm, is explained in the following paragraph. The Python code for it is given in Listing 3.2 and the algorithm is further illustrated in Figure 3.3.

1. Choose an edge as the root. Sort all edges connected to the end point of the root edge according to the zig-zag combing algorithm.

2. Iterate through all of the sorted edges: If the current edge is a non-spanning-tree edge, add it to the path. Otherwise, extend the path by calling this function again with the current edge as the root.

3. Return path.

```
def traverse(root, spanning_tree, pseudoknots):
  path = [["stem", root],]
  neighbors = zig_zag_comb(root)
  for edge in neighbors:
    if edge in pseudoknots:
      path.append(["pseudoknot", edge])
    elif edge in spanning_tree:
      path.extend(traverse(edge, spanning_tree, pseudoknots))
  return path
```

Listing 3.2: Python code for the routing algorithm

3.3 Generation

The generation step places nucleotides on the previously found path. First, we choose a scale-factor, which defines the exact lengths of the edges in nanometers, that is, it determines how many bases are placed on each edge. The factor must be large enough to allow for kissing loops to form. A kissing loop requires at least 13 bases, i.e., the path segment corresponding to such a loop must have room for at least 13 bases. Once a valid scaling factor has been chosen, we can use the default P-stick model parameters to place the beads on the path.

Next, we determine how close to a vertex the helices can be placed to avoid overlaps between helices. This can be thought of as placing such a

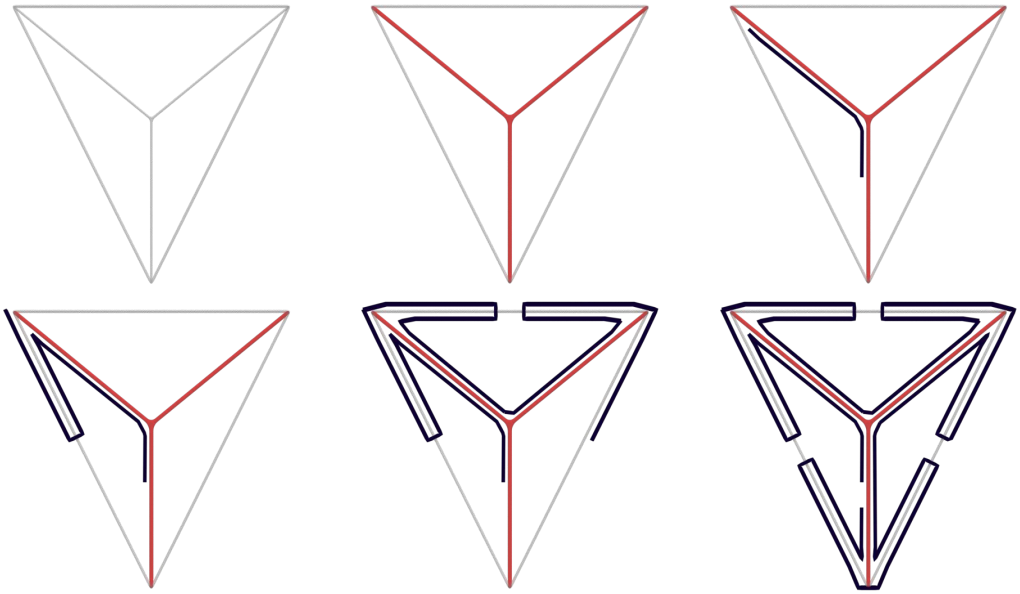


Figure 3.3: The twice around routing algorithm.

sphere on the vertex so that no helix segments outside of it overlap with each other. We do this by using the Formula 3.1 for each vertex, where α is the most acute angle between the edges corresponding to that vertex and r is the radius of the helix in the P-stick model.

$$R = \frac{r}{\tan \alpha} \tag{3.1}$$

As the helices defined by the P-stick model can rotate around their axis, it is not quite enough to know only the path and scale factor, but we must also decide how to orient the helices. Poor orientations might result in long distances between neighboring helices or even in knots. This problem is illustrated in Figure 3.4.

To orient the double helices, they can be modelled as cylinders with four anchors linking them to other helices. Each cylinder will, then, be associated with a four dimensional vector, whose elements consist of the lengths of the links incident to it. Now, the problem of orienting the helices becomes one of minimizing the norm of each of these vectors by rotating the cylinders. The norm of $\|x\|_2$ would minimize the sum of the lengths of links, whereas the norm $\|x\|_\infty$ would minimize the length of the longest link. To do this minimization, we use a spring relaxation algorithm, which tries to iteratively optimize the orientation of each cylinder while the other cylinders stay fixed.

First, we calculate the exact positions of the anchors for each cylinder by

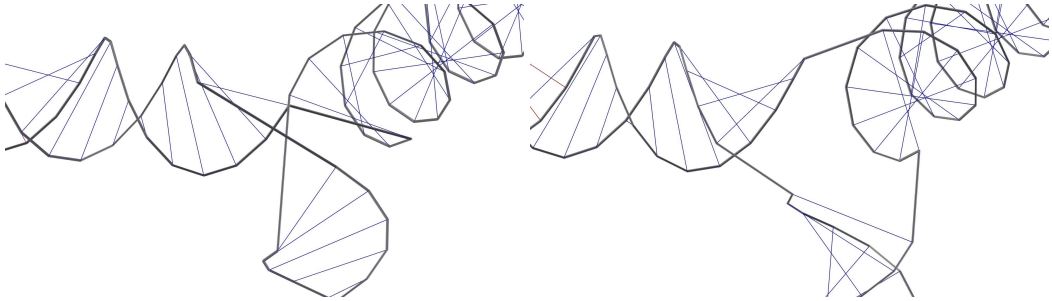


Figure 3.4: An example of a malformed corner and a well formed corner. In the left image all three helices are poorly oriented, resulting in a tangle at the junction. In the second image all helices are well oriented.

using the P-stick model parameters (listed in Table 2.2). The first anchor will be at one end of the cylinder with an angle of 0. The second anchor will be $(num - 1) \cdot rise$ nanometers from first anchor at angle $(num - 1) \cdot twist$, where num is the number of bases on the helix. The third and fourth anchors are the same as the first and second, but they are rotated by $axis$ degrees and shifted by $inclination$ nanometers. The chosen scale factor is very important here, since $twist$ for A-form helices is 32.73°, which means that the angle repeats every 11 steps. Therefore, the anchors on both ends of a cylinder with $11 \cdot x, x \in \mathbb{Z}$ bases will have the same angle. Such cylinders are very easy to orient, and a shape consisting of only them is likely to have a very good optimal solution.

The second step of the spring relaxation is to calculate the torque exerted upon each anchor by the other cylinders. We can calculate this one anchor at a time by projecting the anchor of the other cylinder onto the normal plane of the current cylinder and by calculating the angle between the current position of the anchor and the projection. The equation for this is show in Formula 3.2, where pos is the position of the current anchor, rel the position of the other anchor relative to the current anchor and $helical\_axis$ is the axis of the cylinder. The exact torque is proportional to this angle. We can next add the signed angles of each anchor together to get the total torque exerted upon the cylinder. We then rotate the cylinder accordingly and proceed to the next cylinder.

$$angle = arccos(\frac{pos \bullet (rel - helical\_axis(rel \bullet helical\_axis))}{|rel||helical\_axis|}) \qquad (3.2)$$

Once all the cylinders are optimally oriented, we can proceed by placing the nucleotides on their surfaces as described by the P-stick model. Since the

distance between the nucleotides connecting two anchors together will usually be longer than that between two consecutive bases in a normal helix, we must add extra linkers between helices. This is not a problem, if the number of linkers stays small, i.e., fewer than four, since we would normally add at least one linker between helices anyway in order to allow some flexibility along the joints. If there are too many linkers, however, the structure might start to lose too much of its rigidity.

If a cylinder corresponds to a kissing-loop edge, we generate the nucleotides only half way along the length of it and turn back. The second half is formed as we re-visit the edge later. Because we want the kissing loops to serve the same function as double helices, they must form at a 180 degree angle in relation to each other. This can be enforced by placing two unpaired bases before the loop and one after it as depicted in Figure 3.5 [9]. This also keeps our kissing loops consistent with the P-stick model. Since we use 180 degree kissing loops, we can treat kissing loop edges also as regular cylinders for the spring relaxation phase.

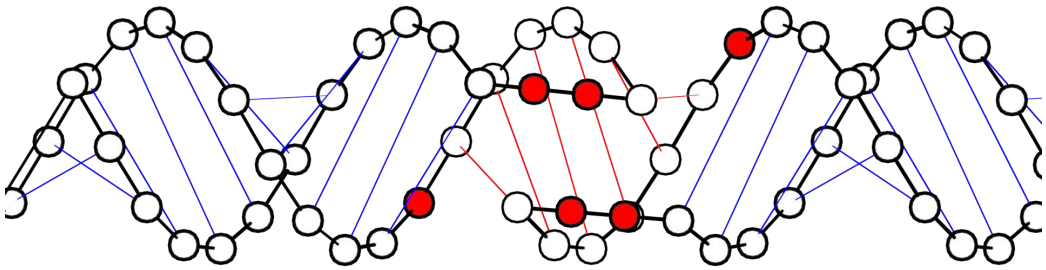


Figure 3.5: A kissing loop. The unpaired bases around the pseudoknot are marked red.

The output of this design process is the secondary and tertiary structures of the RNA strand. The final step is to generate the primary structure.

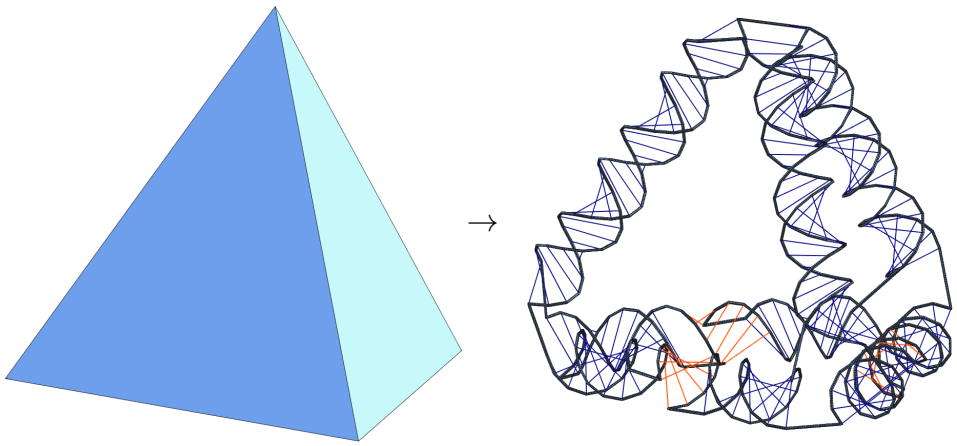
Chapter 4

Generating a sequence

This chapter focuses on generating an RNA primary structure, i.e., the final sequence itself, from its secondary structure. Since primary-structure-generation goes well beyond the scope of this thesis, we use the NUPACK [26] software package for the purpose. NUPACK uses the so-called constrained multistate test tube design algorithm, which optimizes the primary structure based on the nearest-neighbor model.

NUPACK, however, does not support the design of kissing loops. We circumvent this problem by generating kissing loops ourselves and by only generating the stem with NUPACK. This is possible, because NUPACK allows one to fix any number of bases as either specific nucleotides or as any nucleotide from a given class as defined by the nucleic acid notation. Fixing any bases will, of course, shrink the pool of possible primary structures, which might make it impossible to find a suitable primary structure, if the pool is too constrained.

We generate the primary structure in three steps: fix stem bases, find kissing loops and generate the sequence. The whole procedure is demonstrated in Table 4.1. This process is started with the secondary and tertiary structures, which are converted to a primary structure consisting solely of N's in nucleic acid notation, that is, no bases are fixed in any way yet.



↓

Secondary and tertiary structures:

```
..................  ((((((((((((((((( ..  (((( ..[[[[[[.  ))))  ...  ((((((((((((((((( ...  (((( ..[[[[[[.  ))))  ..
(((( ..[[[[[[.  ))))  ..  ))))))))))))))))  ..  ))))))))))))))))  ...  (((( ..]]]]]].  ))))  ...  ((((((((((((((((( ...
(((( ..]]]]]].  ))))  ...  (((( ..]]]]]].  ))))  ...  ))))))))))))))))  ..............
```

↓

Secondary and tertiary structures:

```
..................  ((((((((((((((((( ..  (((( ..[[[[[[.  ))))  ...  ((((((((((((((((( ...  (((( ..[[[[[[.  ))))  ..
(((( ..[[[[[[.  ))))  ..  ))))))))))))))))  ..  ))))))))))))))))  ...  (((( ..]]]]]].  ))))  ...  ((((((((((((((((( ...
(((( ..]]]]]].  ))))  ...  (((( ..]]]]]].  ))))  ...  ))))))))))))))))  ..............
```

Primary Structure with polymerase promoter sequence and linkers fixed (see Section 4.1):

```
GACUAAUACGACUCACUAUA GGGNKNNNNNNNKNNNN AA NNNN NNNNNNNNN NNNN AAA NNNNKNNNNNNNKNNNN AAA NNNN NNNNNNNNN NNNN AA NNNN
NNNNNNNNN NNNN AA NNNNKNNNNNNNKNNNN AA NNNNKNNNNNNNKNCCC AAA NNNN NNNNNNNNN NNNN AAA NNNNKNNNNNNNKNNNN AAA NNNN NNNNNNNNN
NNNN AAA NNNN NNNNNNNNN NNNN AAA NNNNKNNNNNNNKNNNN NNNNNNNNNNNNNNN
```

↓

Secondary structure:

```
..................  ((((((((((((((((( ..  (((( .........  ))))  ...  ((((((((((((((((( ...  (((( .........  ))))  ..
(((( .........  ))))  ..  ))))))))))))))))  ..  ))))))))))))))))  ...  (((( .........  ))))  ...  ((((((((((((((((( ...
(((( .........  ))))  ...  (((( .........  ))))  ...  ))))))))))))))))  ..............
```

Primary structure with kissing loop sequences fixed (see Section 4.2):

```
GACUAAUACGACUCACUAUA GGGNKNNNNNNNKNNNN AA NNUG AAUGCCCCA CGNN AAA NNNNKNNNNNNNKNNNN AAA NNUG AAUGGGCCA CGNN AA NNUG
AAUGGCCCA CGNN AA NNNNKNNNNNNNKNNNN AA NNNNKNNNNNNNKNCCC AAA NNUG AAGGCCCAA CGNN AAA NNNNKNNNNNNNKNNNN AAA NNUG AAGGGGCAA
CGNN AAA NNUG AAGGGCCAA CGNN AAA NNNNKNNNNNNNKNNNN NNNNNNNNNNNNNNN
```

↓

Primary structure with all bases fixed (see Section 4.3):

```
GACUAAUACGACUCACUAUA GGGUGUCGGUCUGCUGC AA GAUG AAUGCCCCA CGUC AAA CCUGGCCUGGGCUCCCG AAA GCUG AAUGGGCCA CGGU AA CGUG
AAUGGCCCA CGUG AA CGGGGGCCUAGGUCAGG AA GCAGUAGACCGAUACCC AAA UCUG AAGGCCCAA CGGA AAA GGAGUGGUACCUGCAAC AAA UCUG AAGGGGCAA
CGGA AAA GGUG AAGGGCCAA CGCC AAA GUUGUAGGUACCGCUCC ACACCCCAAACCCCA
```

Table 4.1: The workflow of converting a mesh to a secondary structure and a primary structure.

4.1 Stem

We define the *stem* of an RNA structure to consist of those bases that form only its secondary structure. This includes all linkers, other unpaired bases and nested base pairs. Only the bases forming any pseudoknots are excluded.

Because RNA strands are transcribed from DNA, some restrictions apply as to what kind of primary structures are feasible. Ideally, the single-stranded DNA used for transcription has no secondary structure at all itself, since that interferes with the process, making it potentially fail completely. Furthermore, synthesizing the DNA strand itself sets additional restrictions. Conventionally, the following sequences are avoided: AAAA, CCCC, GGGG, UUUU, KKKKKK, MMMMMM, RRRRRR, SSSSSS, WWWWWW and YYYYYY [8, supplement p. 11]. The supplier of the DNA template might set further restrictions depending on their synthesis process.

When DNA is transcribed, (T)hymine and (G)uanine transcribe to (U)racil and (G)uanine, respectively. Since U's and G's form strong base pairs but T's and G's repel each other, one can create such an RNA strand that forms secondary structures but whose DNA counterpart does not. By replacing every X:th base pair in the stem with either U-G or G-U, the DNA should have less secondary structure. The ideal value for X being uncertain, the sequences designed in this thesis replace every sixth base pair, rounded up, because that should not be too restrictive in terms of sequence design while still preventing the unwanted DNA secondary structures. In nucleic acid notation, K's correspond to G's and U's, so we can replace every X:th base pair in the stem simply with K's.

As previously defined, the stem does not consist solely of base pairs. There must always be unpaired bases, linkers, between two or more double helices to give the strand enough flexibility to actually form the intended structure at the junctions. One such junction is illustrated in Figure 4.1 Linkers could potentially be any of the four bases as long as they do not pair up with anything. However, since G's and C's form the strongest bonds, it is unlikely that they would not form any base pairs. Therefore, we choose between A's and U's. As U's can form strong bonds with G's, A's might be the obvious choice. However, if there are more than three linkers, we would be violating one of the restrictions by using only A's, so we should instead use a combination of A's and U's. As A's and U's correspond to W's in nucleic acid notation, we will replace linkers with A's when possible and with W's when there are more than three of them consecutively.

To successfully transcribe an RNA strand, it also needs to contain additional *primer* sequences that serve as the starting points in *polymerase chain*

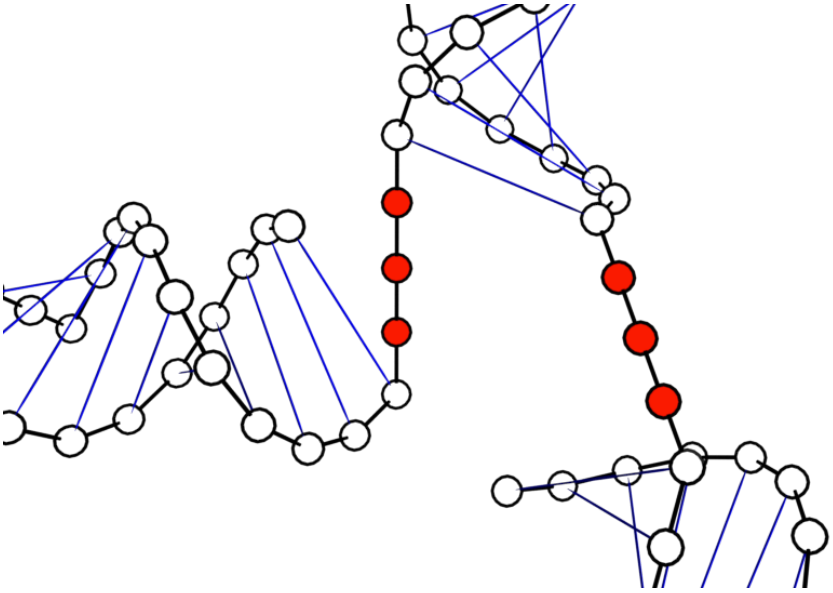


Figure 4.1: A junction between three helices. Linkers are portrayed as red beads.

reaction, a technique used for DNA synthesis. To this end, we add an additional *forward primer* at the beginning, i.e, 5' end, of the primary structure. The exact bases of it are "GACUAAUACGACUCACUAUAGGG" and the matching secondary structure is ".....(((". To the end, we append a *reverse primer*, which is simply fifteen unpaired bases, i.e., the primary structure ends with "NNNNNNNNNNNNNNN" and the secondary structure with ".....".

4.2 Kissing loops

As NUPACK does not consider kissing loops, we must generate them ourselves and then add them to the stem as fixed bases and mark them as unpaired in the secondary structure. This should yield the desired structure, since if a kissing loop were to pair with any bases in the stem, that would violate the modified secondary structure. The problem, then, is to find such loops that they form pseudoknots only with their intended counterparts and not with any of the other loops. This leaves us with two options: Either generate kissing loops completely from scratch or use kissing loops from literature that are known to work well together. Since the number of such kissing loops is a handful at most, we must create new kissing loops for larger structures. The structures in the results chapter all use the kissing

loops extracted from RNAJunction [3] and the 2014 paper by Geary et al. [8, supplement pp. 35–38], but we provide a utility to generate new kissing loops as well, making larger structures at least theoretically possible.

The nearest neighbor model allows us to calculate the energy of any two strands pairing up with each other. If we consider kissing loop base pairs to be energetically essentially the same as regular base pairs, we can use this model to find strong kissing loops that are *orthogonal* to each other. Since we are interested only in the energies of kissing loop base pairs in comparison to other kissing loop base pairs and not to normal base pairs, this model should be adequate for our purposes. As the nearest neighbor model will give us some energy for any two strands pairing up, there is strictly speaking no such thing as orthogonality between two strands. However, in this thesis, we consider two strands to be orthogonal to each other, if their energy is below some threshold in comparison to the strands they are intended to pair up with.

The kissing loops we are using are all six base pairs long, and the loops are flanked by the sequences UGAA on the 5'-side and ACG on the 3'-side. The A's are the padding making sure that the loops are oriented at 180 degree angle, when they form a kissing loop, i.e., they basically form a double helix [9]. The UG and CG initiate the kissing loop, because that maximizies the stability of the loop according to the nearest neighbor model. Since there are six unknown bases in such a loop, there are $4^6 = 4096$ possible loops. A kissing loop consists of two loops, which means that there are $4096^2 = 16777216$ possible pairs in total. This is small enough, so we can actually calculate the energies for all of these. Using NUPACK, again, we calculated the nearest neighbor energetics for all of these strands and saved them in a file in a decreasing order. The first ten lines are listed in Listing 4.1.

```
1  GGGCCG GCCCGG −13.8840164
2  GGCCCG GCCGGG −13.8840164
3  GGGCCG ACCCGG −13.8840163
4  GGCCCA GCCGGG −13.8840163
5  GGCCCA ACCGGG −13.8840163
6  GGGCCA ACCCGG −13.8840163
7  GGGCCA GCCCGG −13.8840163
8  GGCCCG ACCGGG −13.8840163
9  GCCCCA GCGGGG −13.8831708
10 GGGGCG ACCCCG −13.8831708
```

Listing 4.1: The first ten lines of the seventeen million length 6 kissing loops. Note that the energies have been calculated with the A + AA linkers included around the loops even though they are not shown in the listed sequences.

Since we have enumerated all possible kissing loops, we can consider them

all as a weighted complete graph, where loops are nodes and edge weights the energies of each kissing loop. The problem of finding n kissing loops now becomes one of finding such a subgraph of $2n$ nodes, where all edges with weight above some w form a perfect matching of the subgraph, where the matching edges have weights above some T. Finally, maximize for T and minimize for w. Due to time limitations, we must leave this as an open problem. However, we present the following suboptimal algorithm, which should nonetheless be adequate for the purposes of this thesis.

1. Sort all pseudoknots in decreasing order based on their energy.
2. Start reading kissing loops from top down.
3. Calculate the energy difference between both strands of the current pseudoknot and all chosen strands. If they are all higher than X, mark both strands in the pseudoknots as chosen.
4. Return N chosen pseudoknots. If the end of the list is reached, the algorithm has failed.

```
def find_pseudoknots(pair_generator, count):
    chosen = []
    used = set()
    start = next(pair_generator)
    last_energy = -float("inf")
    d = {start[0][0]: [(start[0][1], start[1])], start[0][1]: [(start[0][0], start[1])]}
    for pair, energy in pair_generator:
        if len(chosen) >= count: break
        p = set(d[pair[0]]).union(d[pair[1]])
        conflict = False
        while last_energy - threshold < energy:
            pair_t, energy_t = next(pair_generator)
            last_energy = energy_t
            d.setdefault(pair_t[0], []).append((pair_t[1], energy_t))
            d.setdefault(pair_t[1], []).append((pair_t[0], energy_t))
        for x in p:
            if conflict: break
            if x[0] in used and energy > x[1] - threshold:
                conflict = True
        if not conflict:
            chosen.append(pair)
            used.extend((pair[0], pair[1]))
    return chosen
```

Listing 4.2: An algorithm for finding orthogonal kissing loops.

The algorithm described above will return only orthogonal pseudoknots, since orthogonality is strictly enforced for all chosen strands. However, the result is not necessarily optimal, since only one viable combination of pseudoknots is considered. Moreover, the search might fail even when viable solutions exist. Nevertheless, the algorithm does successfully find dozens of length 6 pseudoknots, when orthogonality is defined as minimum energy difference of 20 % between two strands. Considering how even the most complex structures created in this thesis have only twelve pseudoknots, this should be more than enough. The problem is still an interesting one and might warrant further study.

4.3 Sequence generation

Once all nucleotides are adequately restricted, we can generate the final sequence. We use NUPACK to do this as well, although it is by no means the only software capable of generating RNA sequences. NUPACK will try to find one or more such strands that the number of mispaired bases is below a user defined threshold. If the structure is too constrained, no such sequences exist and NUPACK will naturally fail.

One peculiar problem we encountered while using NUPACK is that the desktop version of it seems to be unable to deal with conflicts between fixed bases and restricted sequences, although the online version works simply by keeping the fixed bases and ignoring the conflicts. Furthermore, we have had some trouble generating sequences of about 1000 nucleotides or longer. It is unclear at this time whether these problems are due to too constrained search pool, too large search pool or something else entirely. However, we have been able to successfully use the software to generate valid sequences for many shapes, including a tetrahedron, a bipyramid and a triangulated prism (Chapter 6).

After the primary structures are generated, we validate them against computer simulations. Tools such as IPknot calculate the minimum nearest neighbor energetics for the structure, producing the expected secondary structure [11]. An example of IPknot output is depicted in Figure 4.2. Other tools, such as Kinefold, SimRNA and oxRNA [4, 23, 24] try to do physical simulations on the actual molecules or coarse-grained representations of them to come up with the folding pathways and the final tertiary structure. Since this latter approach works only for very small structures within reasonable timeframes due to the extremely large search space, it is quite unsuitable for strands longer than some hundreds of nucleotides.

Finally, a validated primary structure can be transcribed from a synthe-

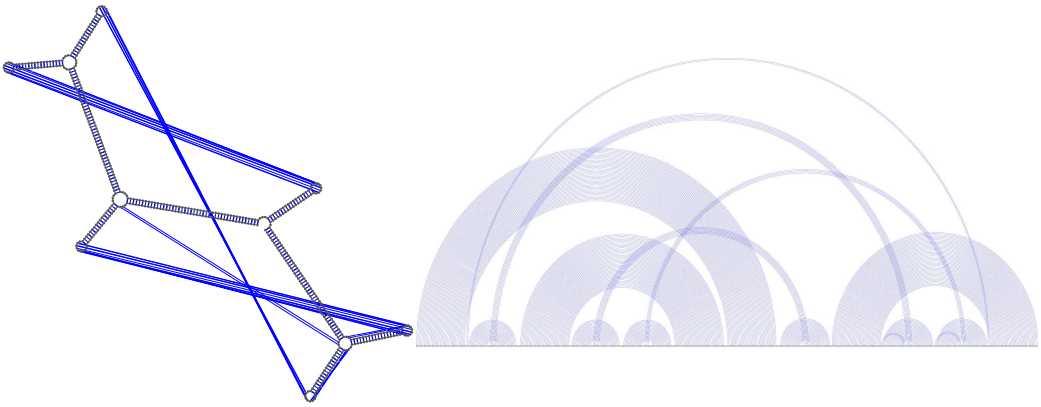


Figure 4.2: An IPknot prediction of the tertiary structure of a tetrahedron.

sized DNA strand, hopefully producing the desired structures in the real world. Since the size of the actual RNA strand will be only some dozens of nanometers, powerful microscopes, such as electron microscopes or atomic force microscopes, will be needed to image them.

Chapter 5

Sterna

Sterna is a tool for converting polyhedra to RNA strands. It is implemented as an add-on to Blender [21], an open-source 3D computer graphics software. At its simplest, a single click generates a random spanning tree, routes a path around it and generates the kissing loops and helices. However, Sterna can also be used interactively to generate a strand one base at a time. Some possible inputs outputs of Sterna are demonstrated in Figure 5.1. Source code for Sterna and additional tools is available from `https://github.com/Ritkuli/sterna`.

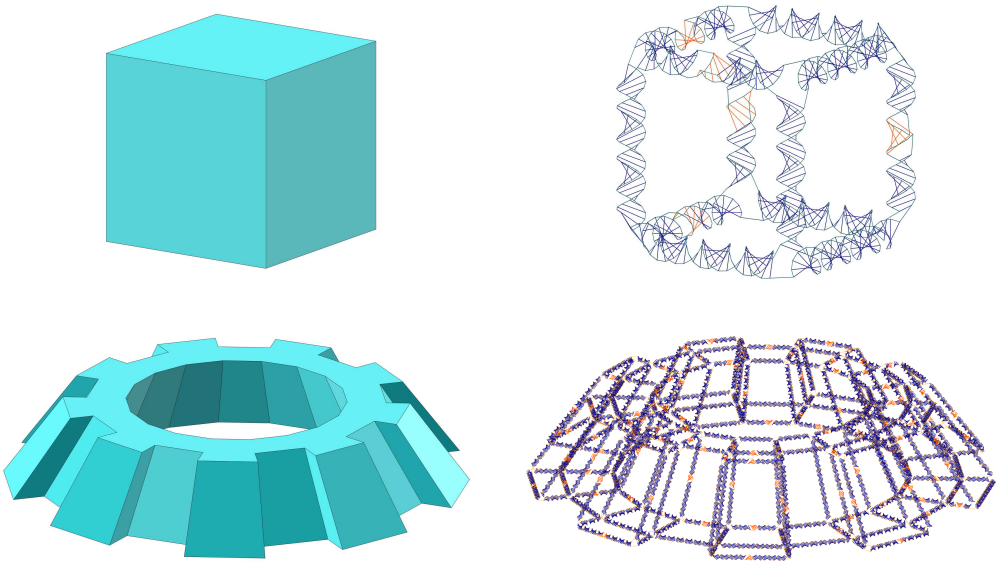


Figure 5.1: Typical use-cases of converting 3D-models into RNA strands.

5.1 Automatic generation

Sterna can be used to automatically convert a 3D-model into a viable RNA structure based on various parameters as shown in Figure 5.2. By using a random spanning tree, the user only has to define the scale, and Sterna will convert the selected mesh into a strand. The user is naturally free to manually select the spanning tree edges as well. The scale-parameter defines how many nanometers each unit of length in Blender is. Alternatively, the user may choose to only define how many full turns a helix should be, and Sterna will automatically chooose the appropriate scale. A polyhedron can, of course, have many edges with different lengths, making it impossible for all of the helices to be the same length. Therefore, this parameter only guarantees that the most common edge length is equivalent to the given number of turns. Other available parameters include minimum and maximum number of linkers between helices, a switch for spring relaxation and the length that should be culled from all edges at vertices. The adaptive offset switch can be toggled for Sterna to calculate this culling on an edge-by-edge basis.

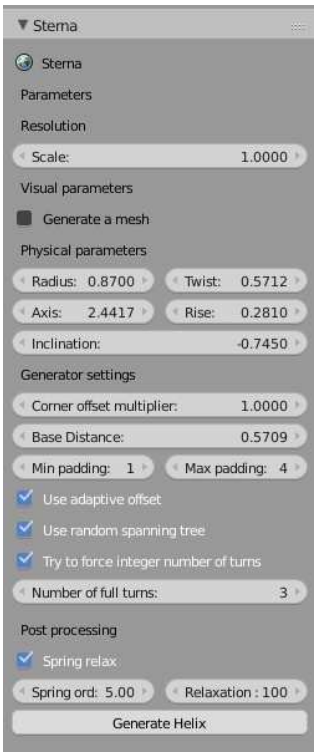


Figure 5.2: Generator parameters.

Sterna also allows the user to manipulate the parameters of the P-stick model. Theoretically, this should allow Sterna to also work with DNA or even artificial bases.

The input to Sterna is any single polyhedron. Only single polyhedra are accepted, since the structure is traced with just one RNA strand. A future version might feature an option to allow multiple polyhedra and therefore to generate multiple strands without having to split the 3D-model into its connected subcomponents. Since Blender supports most 3D-model file formats, Sterna is able to turn almost any model into an RNA strand.

By generating a new strand, Sterna creates a new mesh, where vertices represent bases and edges represent connections between them. The connections can either be backbone-connections, base pairs or pseudoknots. The user can then manually modify this mesh or export it as is.

5.2 Interactive generation

Sterna was not designed with interactivity in mind, but it has a rudimentary support for that as well. Since Sterna generates a normal mesh, it can also be edited just like any other with certain restrictions. The user is free to move, add and delete bases. Pairing bases as either A-helix stem or kissing loops is also possible. A simple edit operation is depicted in Figure 5.3.

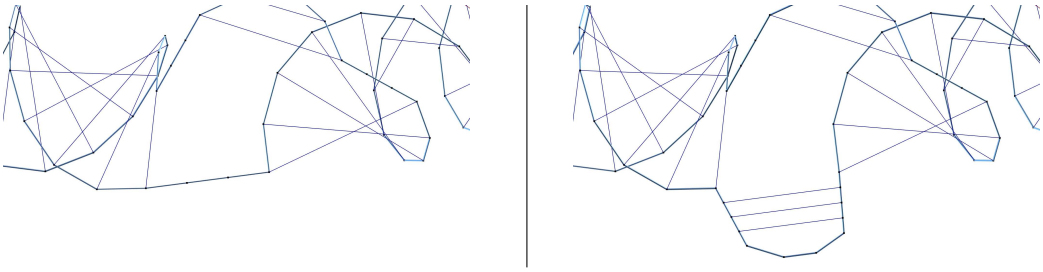


Figure 5.3: Manually adding a new loop.

There are no checks in place to prevent the user from creating infeasible structures. For instance, the user may create cycles, add multiple base pairs to one base or even branch the backbone multiple times. The order of the bases is determined by Blender's internal indexing, so some of the more advanced operations might also break the structure. These limitations might prevent some strands from being exported. However, by fixing the problems or by undoing the changes, the user is again able to export the structure.

In theory, it would be possible to create the whole strand manually, although this would be extremely cumbersome, since Sterna has no functionality to add full helices manually at this time. Future versions might include helper functions to produce full helices, kissing loops and multi-loops, but no such features exist at the time of writing this thesis.

5.3 Output

Sterna itself will generate a Blender mesh with such connectivity that it can be interpreted as an RNA strand. This mesh can be saved as a normal Blender blend file or the user may export it in a variety of 3D-model file formats. However, Sterna also provides an export-option to a snac-file, which is the intended output for Sterna. Sterna is also able to import snac files, if the user wishes to modify the structure or to simply visualize it again.

A snac file is a simple text file with a notation based on the YAML markup language [2]. The name is an acronym for Simple Nucleic Acid Configuration. A snac file contains fields and values separated by colons. A field can be any string, and the values can either be single strings or multiple strings separated by commas. Comments are marked with hash tags. An example snac file is shown in Listing 5.1.

By default, Sterna will export a snac file with primary structure, secondary structure, secondary structure numbering and coordinates fields. The primary structure is a string based on the nucleic acid notation. Secondary structure is a dotbracket-string with additional square brackets for pseudoknots. The secondary structure numbering field indexes the pseudoknots to help avoid ambiguity, if kissing loops happen to cross. This, however, is mostly redundant, since Sterna outputs only structures where kissing loops are nested, unless the user manually alters them to be crossing. Finally, the positions field lists the exact coordinates for all bases in the RNA strand in nanometers.

```
# Nucleic acid notation  '|':
# NNNKNNNNNKNNNNNKNNNNNKNNNNNKNNNNN | A |
    NNNKNNNNNKNNNNNKNNNNNKNNNNNKNNNNN | A | NNNKNNNNNKNN |
    NNNNNNNNN | NNKNNNNNKNNN | AAA |
    NNNNNKNNNNNKNNNNNKNNNNNKNNNNNKNNN | AAA |
    NNNNNKNNNNNKNNNNNKNNNNNKNNNNNKNNN | AAA |
    NNNKNNNNNKNNNNNKNNNNNKNNNNNKNNNNN | AA | NNNKNNNNNKNN |
    NNNNNNNNN | NNKNNNNNKNNN | AA |
    NNNNNKNNNNNKNNNNNKNNNNNKNNNNNKNNN
primary_structure:
    NNNKNNNNNKNNNNNKNNNNNKNNNNNKNNNNNANNNKNNNNNKNNNNN
    KNNNNNKNNNNNKNNNNNANNNKNNNNNKNNNNNNNNNNNNNKNNNNNK
    NNNAAANNNNNKNNNNNKNNNNNKNNNNNKNNNNNKNNNAAANNNNNKN
    NNNNKNNNNNKNNNNNKNNNNNKNNNAAANNNKNNNNNKNNNNNKNNNN
    NKNNNNNKNNNNNAANNNKNNNNNKNNNNNNNNNNNNNKNNNNNKNNNA
    ANNNNNKNNNNNKNNNNNKNNNNNKNNNNNKNNN
# domains separated by |:
# ((((((((((((((((((((((((((((((((( | . |
    ((((((((((((((((((((((((((((((( | . | ((((((((((( |
    ..[[[[[[. | )))))))))))) | ... | ))))))))))))))))))))))))))))
```

```
    ))))) | ... | ))))))))))))))))))))))))))))))))) | ... |
    ((((((((((((((((((((((((((((((( | .. | ((((((((((( |
    ..]]]]]]. | )))))))))) | .. | ))))))))))))))))))))))))))))
    ))))
secondary_structure:
    ((((((((((((((((((((((((((((((.((((((((((((((
    ((((((((((((((((.((((((((((..[[[[[[.)))))))) )))...)))))
    ))))))))))))))))))))))))))...)))))) ))))))))))))))))))))
    ))))...((((((((((((((((((
    ((((((((((((..((((((((((((..]]]]]].)))))))))))). .))))))))))
    ))))))))))))))))))))
# the following numbers are assigned to pseudoknot complexes in
    the same order as they appear in the secondary structure. For
     instance, 1 2 1 2 would mean that the first complex connects
     to the third one, and the second one connects to the fourth.
pseudoknot_numbering: 0 0
# The coordinates are represented as comma-separated three-
    tuples, x y z, in nanometers. The coordinates correspond to
    the primary and secondary structure in the same order.
positions: [-4.833355903625488 3.9717981815338135
    0.4350000023841858, -5.188178062438965 3.690798044204712
    0.7733058929443359, -5.669579982757568 3.4097981452941895
    0.8660528063774109, -6.124694347381592 3.128798246383667
    0.6837893724441528, -6.409002780914307 2.8477981090545654
    0.2843928039073944, -6.432225704193115 2.566798210144043
    -0.20531120896339417, -6.18698787689209 2.2857983112335205
    -0.6298201680183411, -5.751162052154541 2.004798173904419
    -0.8543331623077393, -5.263144493103027...
```

Listing 5.1: An example snac file

5.4 External tools

Sterna is complemented with three additional tools, snacseq, pkgen and snac2ox. They extend the functionality of Sterna outside the framework of Blender. All of them are commandline software implemented with Python.

5.4.1 Snacseq

Snacseq is a commandline tool we developed to generate the primary structure of an RNA strand based on its secondary structure and various other constraints. It uses Pkgen to select pseudoknots, and it generates the primary structure with NUPACK. The input is a snac-file, and the output either modifies the input file or generates a new snac-file.

Snacseq generates the input for NUPACK by completing a premade template file, which contains fields such as prevented sequences, temperature and sodium content. Snacseq also allows the user to define the maximum GC-content, to enforce certain pseudoknots to be selected or to define the minimum energy between mismatching pseudoknots. Snacseq can also be used to generate only pseudoknots, if the user wishes to generate the rest of the sequence by other means.

5.4.2 Pkgen

Pkgen is a tool used to generate and select pseudoknots. When used as a commandline program, it generates all possible combinations of two strands of length N and calculates the nearest-neighbor-energy of them using NUPACK. It takes N as an input and outputs a sorted list of the strands and their energies. Pkgen also implements an API of Python functions that can be called directly from another Python script to select pseudoknot strands from a list based on restrictions such as GC-content, prevented sequences or the energy between mismatching strands.

5.4.3 Snac2ox

Snac2ox is a preliminary converter used to convert snac-files to the input format of oxRNA [23], i.e., a topology file, a configuration file and a simulation setup file. It also allows the user to generate a file describing kinetic traps between either pseudoknot base pairs or all base pairs.

Since Sterna is based on the quite coarse-grained P-stick model, and because it inserts various linkers in the strand, the distances between bases are not necessarily exactly correct. Therefore, as we have found oxRNA to be very sensitive to such errors, snac2ox also runs the relaxation simulation shipped with oxRNA, which tries to correct them. The converter can, of course, be run without the relaxation, but the simulation is unlikely to work without it.

Chapter 6

Preliminary results

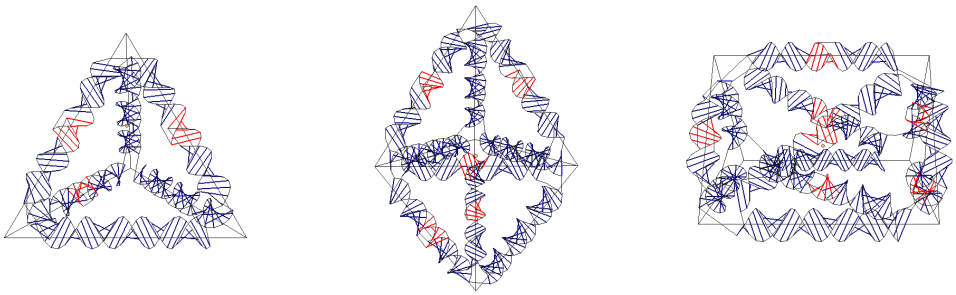


Figure 6.1: Three designs with Sterna from left to right: a tetrahedron, a bipyramid and a triangulated prism.

Using the tools and processes described in the previous chapters, we designed three RNA strands for the target polyhedra of a tetrahedron, a bipyramid and a triangulated prism as depicted in Figure 6.1. The secondary structures were designed with Sterna, kissing loops were chosen from a list of known pseudoknots, listed on Table 6.1, and the primary structures were generated using NUPACK. The DNA templates were ordered from IDT,[1] and the final RNA strands were synthesized from in vitro transcription of the DNA templates by our collaborators Ibuki Kawamata and Lukas Oesinghaus.

[1]Integrated DNA Technologies. https://eu.idtdna.com/pages

GGAGGC CCUCCG
GUGGAC CACCUG
GCCUGC CGGACG
GCGAGC CGCUCG
AGGCUC UCCGAG
GCGUUC CGCAAG
GUCACC CAGUGG
CGUGGU GCACCA
CUUCGC GAAGCG

Table 6.1: The list of known kissing loops. Extracted from RNAJunction [3] and the 2014 paper by Geary et al. [8, supplement pp. 35–38].

The complete primary and secondary structures for the tetrahedron, bipyramid and prism are listed in Appendix A in Tables A.1.1, A.1.2 and A.1.3, respectively. The tetrahedron is 455 nucleotides long, the bipyramid 663 nt long and the prism 801 nt long.

6.1 Computer simulations

Since the designed strands are relatively long, the smallest being well over 400 nucleotides long, it would presumably be very time consuming to simulate their folding process using typical molecular dynamics tools. Therefore, only minimum free energy calculations are available at this time. To this end, we used IPknot, which tries to predict minimum free energy conformations using the nearest neighbor model.

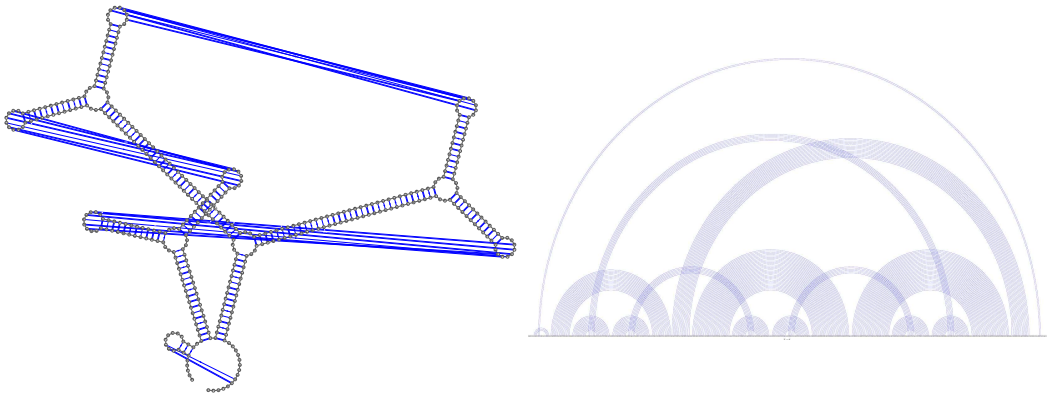


Figure 6.2: IPknot prediction for the structure of the tetrahedron.

The IPknot results for the tetrahedron are shown in Figure 6.2, the results for the bipyramid in Figure 6.3 and the results for the prism in Figure 6.4. The stem base pairs, i.e., the secondary structures, seem to be very well formed in all of the predicted structures. The tetrahedron has a small defect at the 5'-end, the bipyramid has a slightly malformed first kissing loop, and the prism has a similar deformity in the third kissing loop. The predicted tertiary structures are more worrisome.

The tetrahedron's tertiary structure is nearly perfect (Figure 6.2). This is to be expected, since it is the simplest of the designs and contains only three pseudoknots. There are only two mispaired pseudoknot bases located at the defective part of the secondary structure. When the three dimensional geometry is considered, one can reasonably expect the tetrahedron to fold properly.

The bipyramid (Figure 6.3) is decently well formed, although it misses one pseudoknot completely. This might be due to the deformity in the secondary structure of the first kissing loop. There is also some mispairing between the second and third kissing loops. Again, the three dimensional geometry might still pressure the defective parts to fold as designed.

The predictions for the triangulated prism (Figure 6.4) are the grimmest.

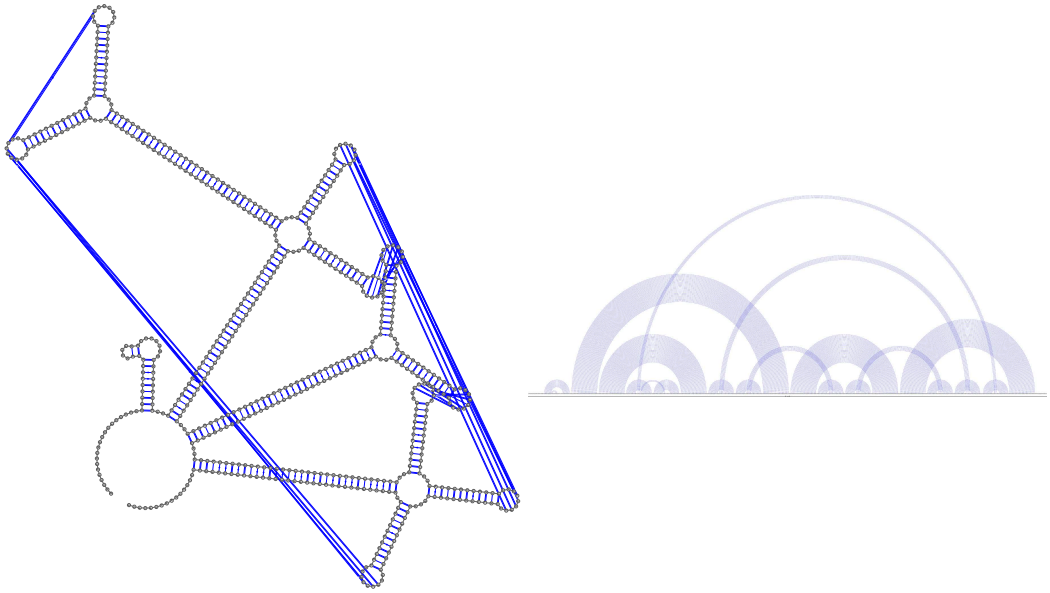


Figure 6.3: IPknot prediction for the structure of the bipyramid.

It has two missing pseudoknots with the first kissing loop forming a pseudoknot with the stem. It is the largest and most complicated of our structures, so these defects are to be expected. Five out of seven pseudoknots are well-formed, however, so there is a chance the structure might fold properly in nature.

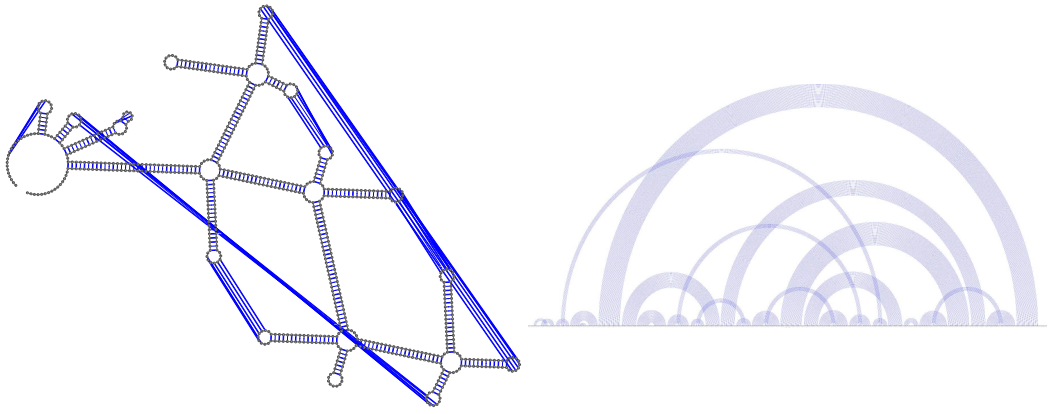


Figure 6.4: IPknot prediction for the structure of the prism.

6.2 Laboratory results

Due to time constraints, the electron microscope results for all of the designs are incomplete. The DNA templates for each structure were acquired and transcribed into RNA, but we were unable to perfrom the final imaging with a high resolution microscope due to problems with logistics. We were able to acquire low-quality atomic force microscope images only for the tetrahedron and the prism. Nevertheless, the simulations for all of the structures are encouraging, and the preliminary images for the tetrahedron and prism are very promising. The images were taken by our collaborator Ibuki Kawamata.

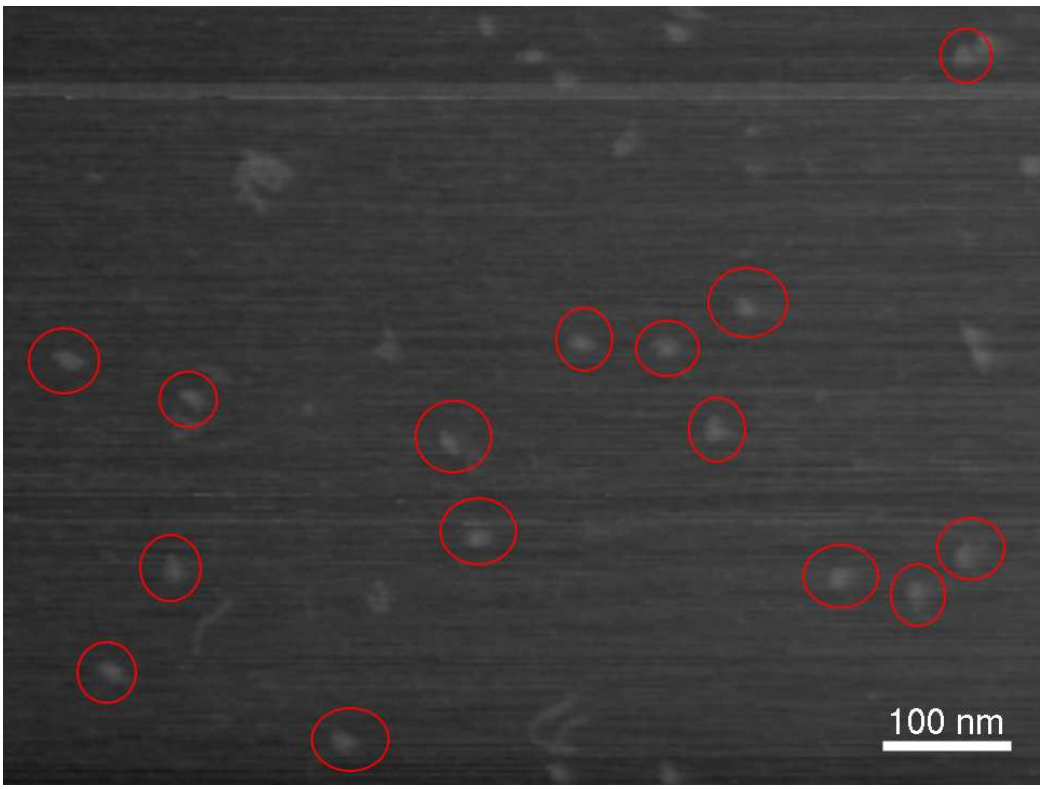


Figure 6.5: An atomic force microscope image of ostensibly well folded RNA tetrahedra.

AFM images for the tetrahedra are shown in Figures 6.5, 6.6 and 6.7. In the first image, we can see numerous tiny specks and longer string-like shapes saturating the image. We expect the strings to be unfolded RNA strands or RNA-DNA hybrids, but the tiny specks have dimensions roughly matching those of the designed tetrahedron. The dimensions for the bounding box of the designed tetrahedron are 12.5 nm x 11.2 nm x 10.3 nm. The yield

of seemingly well formed structures might be a result of improper folding conditions or the stresses of AFM imaging.

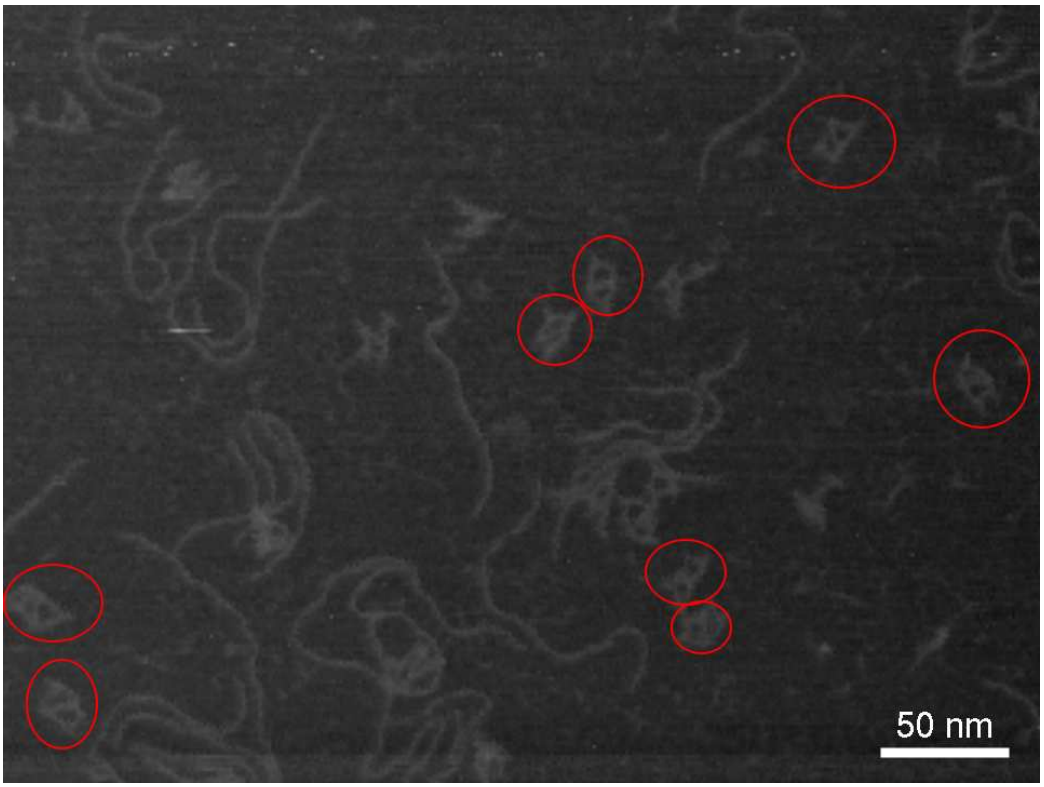


Figure 6.6: An atomic force microscope image of partially folded tetrahedra.

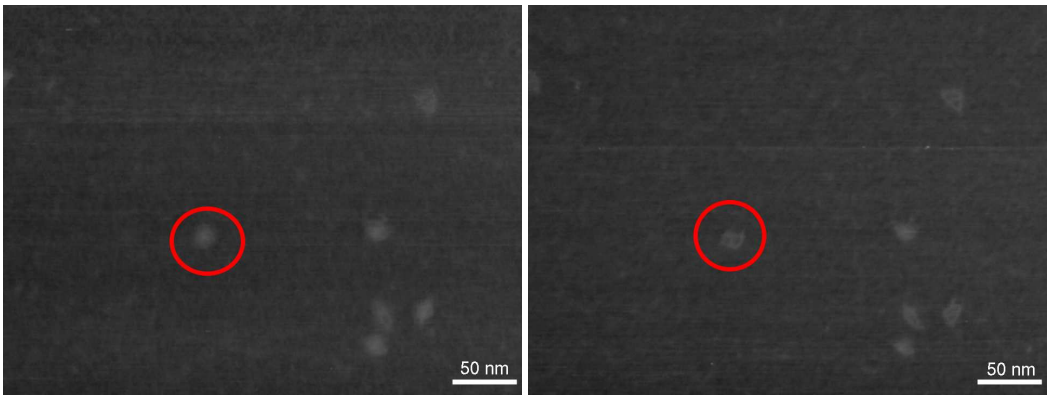


Figure 6.7: Atomic force microscope images of a tetrahedra seemingly unfolding.

The image in Figure 6.6 seems to contain numerous double-triangles

alongside the unfolded RNA. We expect these to be partially formed tetrahedra, where one kissing loop has not formed. The tetrahedra might have either broken open or they might have never formed in the first place. The images in Figure 6.7 in particular seem to depict a kissing loop of a tetrahedron actually breaking up. These partially formed structures do lend credibility to the claim that the specks in Figure 6.5 are indeed properly folded tetrahedra.

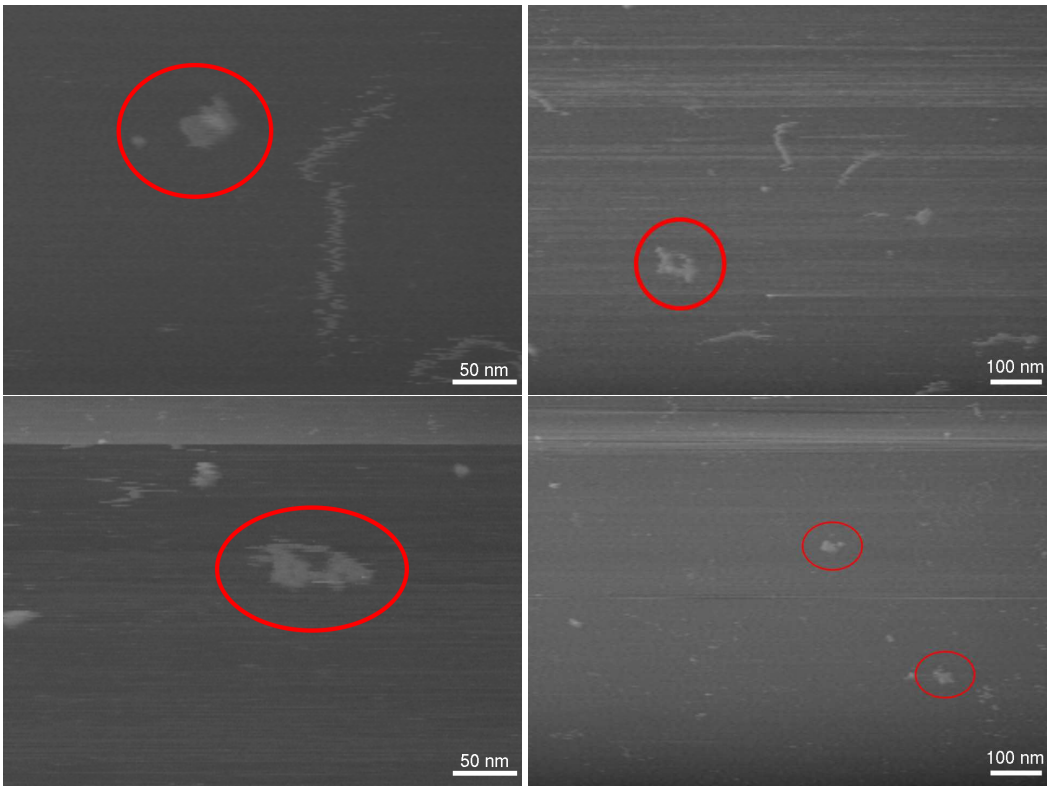


Figure 6.8: Atomic force microscope images of a triangulated prism.

The images in Figure 6.8 depict our triangulated RNA prisms. These results are somewhat less exciting. It is hard to say anything definite at this time, since the number of seemingly well-folded structures is low, and even they are quite obscure. The dimensions for the bounding box of the designed prism are 11 nm x 14 nm x 9.7 nm. The low yield is again likely due to either the folding process or the sample being contaminated with DNA. We must wait for higher resolution images of the prism before making claims about how well or how poorly they are folded. This is of course true for the tetrahedra as well.

Chapter 7

Conclusions

In this thesis, we have studied and expanded upon RNA-based bottom-up nanotechnology. We have introduced the problem of designing polyhedra from single-stranded RNA. We provide a solution to it with a scheme based on spanning trees and the elementary motifs of kissing loops and double helices.

In Chapter 3, we showed how to route a strand around any spanning tree without forming topological knots. We introduced the zig-zag combing algorithm and we explained how we can generate RNA strands based on the P-stick model by Geary and Andersen. Chapter 4 described how we generate our primary structures. We also presented a method for generating sets of orthogonal kissing loops. Chapter 5 introduced Sterna, explained its capabilities and introduced the snac file format. We also presented three additional scripts to generate pseudoknots and primary structures and to convert RNA strands from snac format to oxRNA. Finally, in Chapter 6, we discussed how we designed and simulated three single stranded RNA polyhedra. These designs were synthesized and imaged elsewhere, as discussed in Section 6.2.

Possible follow-up works of this thesis should focus on refining and improving the routing algorithm and the spring relaxation introduced in Chapter 3 Section 3.3 by optimizing the length of the cylinder in addition to the orientation. Another interesting problem would be the search for orthogonal kissing loops, which would involve determining the exact free energies for kissing loops and solving the sub-graph problem introduced in Chapter 4 Section 4.2. Further work on Sterna (Chapter 5) could focus on improving its usability and interactivity and on adding support for more diverse secondary structure motifs.

In conclusion, we have introduced a fully automated design process of converting almost arbitrary polyhedra into RNA strands. Our results give credence to its viability. Even so, we have introduced new problems and the

process is far from perfect. By working on refining Sterna and by solving the problems encountered, we might be on our way to be able to design and synthesize truly arbitrary nano-scale objects with RNA.

Bibliography

[1] Abels, J., Moreno-Herrero, F., Van der Heijden, T., Dekker, C., and Dekker, N. H. Single-molecule measurements of the persistence length of double-stranded RNA. *Biophysical Journal 88*, 4 (2005), 2737–2744.

[2] Ben-Kiki, O., and Evans, C. Yaml. `http://yaml.org/`, 2018. [Online; accessed 2018-08-19].

[3] Bindewald, E., Hayes, R., Yingling, Y. G., Kasprzak, W., and Shapiro, B. A. RNAJunction: a database of RNA junctions and kissing loops for three-dimensional structural analysis and nanodesign. *Nucleic Acids Research 36*, suppl_1 (2007), D392–D397.

[4] Boniecki, M. J., Lach, G., Dawson, W. K., Tomala, K., Lukasz, P., Soltysinski, T., Rother, K. M., and Bujnicki, J. M. SimRNA: a coarse-grained method for RNA folding simulations and 3D structure prediction. *Nucleic Acids Research 44*, 7 (2015), e63–e63.

[5] Bricard, R. Mémoire sur la théorie de l'octaèdre articulé. *Journal de Mathématiques pures et appliquées 3* (1897), 113–148.

[6] Connelly, R. A counterexample to the rigidity conjecture for polyhedra. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques 47*, 1 (1977), 333–338.

[7] Darnell, J. E., Lodish, H. F., Baltimore, D., et al. *Molecular Cell Biology*, vol. 2. Scientific American Books New York, 1990.

[8] Geary, C., Rothemund, P. W., and Andersen, E. S. A single-stranded architecture for cotranscriptional folding of RNA nanostructures. *Science 345*, 6198 (2014), 799–804.

[9] Geary, C. W., and Andersen, E. S. Design principles for single-stranded RNA origami structures. In *International Workshop on DNA-Based Computers* (2014), Springer, pp. 1–19.

[10] Gibson, D. G., Benders, G. A., Andrews-Pfannkoch, C., Denisova, E. A., Baden-Tillson, H., Zaveri, J., Stockwell, T. B., Brownley, A., Thomas, D. W., Algire, M. A., et al. Complete chemical synthesis, assembly, and cloning of a Mycoplasma genitalium genome. *Science 319*, 5867 (2008), 1215–1220.

[11] Kato, Y., Sato, K., Asai, K., and Akutsu, T. Rtips: fast and accurate tools for RNA 2D structure prediction using integer programming. *Nucleic Acids Research 40*, W1 (2012), W29–W34.

[12] Leontis, N., and Westhof, E. *RNA 3D structure analysis and prediction*, vol. 27 of *Nucleic Acids and Molecular Biology*. Springer Science & Business Media, 2012.

[13] Mohammed, A., Orponen, P., and Pai, S. Algorithmic design of cotranscriptionally dolding 2D RNA origami structures. In *International Conference on Unconventional Computation and Natural Computation* (2018), Springer, pp. 159–172.

[14] Murty, B., Shankar, P., Raj, B., Rath, B., and Murday, J. *Textbook of Nanoscience and Nanotechnology*. Springer Science & Business Media, 2013.

[15] Orponen, P. Design methods for 3D wireframe DNA nanostructures. *Natural Computing 17*, 1 (2018), 147–160.

[16] Rothemund, P. W. Folding DNA to create nanoscale shapes and patterns. *Nature 440*, 7082 (2006), 297.

[17] Russell, R. *Biophysics of RNA Folding*, vol. 3. Springer Science & Business Media, 2012.

[18] Saenger, W. *Principles of Nucleic Acid Structure*. Springer Science & Business Media, 2013.

[19] Schelle, B., and Thiel, V. T7 RiboMAX Express: Generation of 27kb in vitro Transcripts in Minutes. `http://fi.promega.com/resources/pubhub/enotes/t7-ribomax-express-generation-of-27kb-in-vitro-transcripts-in-minutes/`, 2002. [Online; accessed 2018-08-12].

[20] Schleif, R., et al. *Genetics and Molecular Biology.* No. Ed. 2. Johns Hopkins University Press, 1993.

[21] The Blender Foundation. Blender. `http://www.blender.org/`, 2018. [Online; accessed 2018-08-02].

[22] Veneziano, R., Ratanalert, S., Zhang, K., Zhang, F., Yan, H., Chiu, W., and Bathe, M. Designer nanoscale DNA assemblies programmed from the top down. *Science 352*, 6293 (2016), 1534–1534.

[23] Šulc, P., Romano, F., Ouldridge, T. E., Doye, J. P. K., and Louis, A. A. A nucleotide-level coarse-grained model of RNA. *The Journal of Chemical Physics 140*, 23 (2014).

[24] Xayaphoummine, A., Bucher, T., and Isambert, H. Kinefold web server for RNA/DNA folding path and structure prediction including pseudoknots and knots. *Nucleic Acids Research 33*, suppl_2 (2005), W605–W610.

[25] Xia, T., SantaLucia Jr, J., Burkard, M. E., Kierzek, R., Schroeder, S. J., Jiao, X., Cox, C., and Turner, D. H. Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson- Crick base pairs. *Biochemistry 37*, 42 (1998), 14719–14735.

[26] Zadeh, J. N., Steenberg, C. D., Bois, J. S., Wolfe, B. R., Pierce, M. B., Khan, A. R., Dirks, R. M., and Pierce, N. A. NUPACK: analysis and design of nucleic acid systems. *Journal of Computational Chemistry 32*, 1 (2011), 170–173.

Appendix A

Appendix

A.1 Designs

A.1.1 Tetrahedron

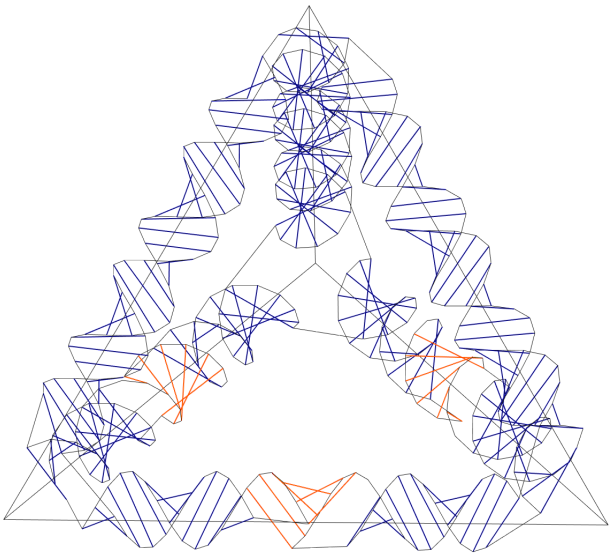


Figure A.1: The Sterna design of the tetrahedron.

```
Primary structure:
GACUAAUACGACUCACUAUAGGGUCGGCGUCUCAGUAAAUCGCGGAGGCUGAAGGAGGCACGGCCUC
UGCGAAACUGGGCGAGGUGAAGCGAGCACGCCUCGUCCAGAAUACUGGGACGCUGACCCGGACUUCC
GGUUCCGAAACAGCGUUUCUAGGCGUCUCGGAGCCAUAUGGCGAACCUGUCCUGGUGAAGCUCGCAC
GCCAGGGCAGGAACCGUGCUCGGUGAAGUGGACACGCCGAGUACGGAACGCCAUGUGGCUCUGAGAC
GUCUAGAAGCGCUGAAGUGCAUCAGUUUGACGCUGUGCAUAUGGCACAGAACGACUGGAGGUGAAGU
CCACACGCCUCCGGUCGAACGACUGAGGGUGAAGCCUCCACGCCCUCGGUCGAACUGUGCUAUAUGC
GCAGCGUUAAACUGGUGCACAAUCGGAGCCGGAGGUCCACUUCACUUCACUCC
Secondary structure:
.....................(((((((((((((((((..((((((((((..[[[[[[.)))))))
)))))..((((((((((..]]]]]].))))))))))..)))))))))))))))))(((((((((
(((((((..(((((((((((((((((((((((((((((..((((((((((..[[[[[[.)
))))))))))..((((((((((..]]]]]].))))))))))..))))))))))))))))))
))))))))))))..(((((((((((((((((((((((((((((..((((((((((..[[
[[[[.))))))))))))..((((((((((..]]]]]].))))))))))..))))))))))))
))))))))))))))))))..))))))))))))))))..............
```

Table A.1: The tetrahedron.

A.1.2 Bipyramid

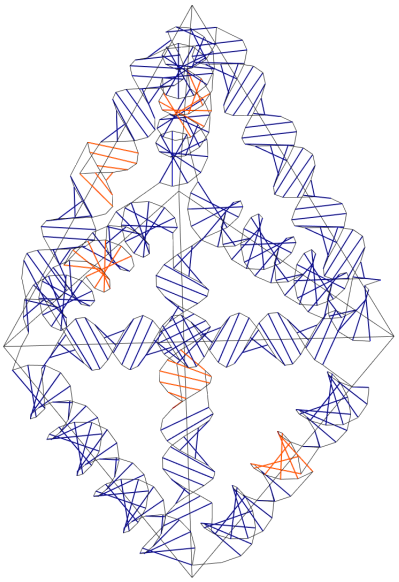


Figure A.2: The Sterna design of the bipyramid.

```
Primary structure:
GACUAAUACGACUCACUAUAGGGUCUACGCUGAAAGGCUCACGGCGUAGGCCCAAGCUUGGGAUGAU
GUCUGACAAGUUCUAUCCAGGAACCGUGUACGGCCAUCGAGUAAUAGAGCUACUGCAAGGCUCAGAG
GUGAAGAGCCUACGCCUCUGGGCCAAGCUGGGUCGGUGAAGCGAGCACGCCGACCUAGCAAGCAGUG
GCUCUGUUACUUGAUGGUCGUACGCGGAAGCGUGAGGUCUGAAGCCUGCACGGACCUCGCGCAACGG
UGAUCGGUGAAGUGGACACGCCGAUCGCCGAACCUGGGUAGAAUUUGUCGGACAUUAUCCCGAGCAA
GGGUCAGCUGCCAGGGUCAGGUGAUUCUGCCUGAAACCGGAGACCUGAAGUCCACACGGGUCUCUGG
UAAACGGGCUCGGUGAAGGAGGCACGCCGAGCUCGUAACAGGCGGAAUCGCCUGAUCCUGGUAGCUG
GCCCAAGGCUUAGAGUUACAAGUUGAGGUCUACUAAUCCAAGGGUACAGGGUGAAGCCUCCACGCCC
UGUGCCCAAGACGGGCUGGUGAAGCAGGCACGCCAGCCUGUCAAAGGCGGUCUGGUGAAGCUCGCAC
GCCAGACUGCCAGGAUUGGUAGAUCUCAAUUUGUAGCUCUAGGCCACUUCUAACUACACA
Secondary structure:
.....................(((((((((((..[[[[[[.)))))))))))..((((((((((((
((((((((((((((((((((..((((((((((((((((((((((((((((((..(((((((((
(((..]]]]]].))))))))))))..((((((((((((..[[[[[[.))))))))))))..))))))
))))))))))))))))))))))))..((((((((((((..[[[[[[.))))))))))))..(((
(((((((((..[[[[[[.))))))))))))..)))))))))))))))))))))))))))))))..
((((((((((((((((((((((((((((((..((((((((((((..]]]]]].))))))))))))
)..((((((((((((..[[[[[[.))))))))))))..))))))))))))))))))))))))))
))))..((((((((((((((((((((((((((((((..((((((((((((..]]]]]].)))))
)))))))..((((((((((((..]]]]]].))))))))))))...((((((((((((..]]]]]].)
))))))))))).)))))))))))))))))))))))))))))))).............
```

Table A.2: Bipyramid.

A.1.3 Prism

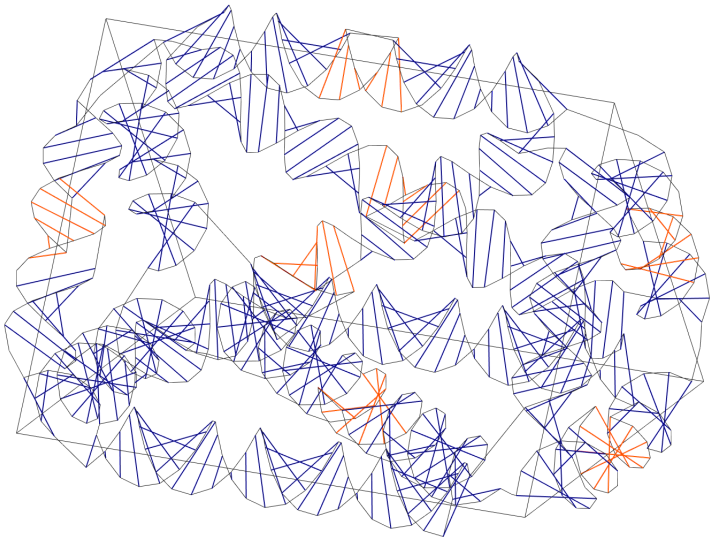


Figure A.3: The Sterna design of the triangulated prism.

Primary structure:

```
GACUAAUACGACUCACUAUAGGGAUGAAGUCACCACGUCCCAAGGUCUGAAGCGUUCACGGACCAGC
AAGUGUGCGGCACUGAAAGGCUCACGGUGUCGCACAUUUGCAGCGUGUGCGGUGUAAUAGCGUGAUG
CAUUGAGCAAAGGCAGUACAUUUGCUUUGUACCAGGGUACGUUGCGGUCGUGAAGAGCCUACGCGAC
CGUAACGUGCCCAAGCGUGACCUCUGAAGCGAGCACGGAGGUCGCGCAAAGUCCUGAAGCCUGCACG
GGACAAGGUGCAAAGCGAAUGUAUUGCCAACGGUCACCGAUUAUCCAUGCCAAGGUACUGAAGCAGG
CACGGUACCAGGAAUUGGAUACGGGUGAAGUGGACACGCCUGUAUCCAGUUCCAGACUCGGACGUUC
CGGCAGCAUAGGCAUAUCCGAAAGCUAUCUGCCUGCGAGCUUGGCAGCGUAGCCUUUACUGGUGAAG
UCCACACGCCAGUAGAGGCUGCGCAACGGUACUCGGUGAAGCUCGCACGCCGAGUGCCGAAAGCGUC
UGAAGAACGCACGGACGCAGCCGAGCUCGUAGGCAGGUAGCAACAGGGUGAAGGUGACACGCCCUGA
GGACGUGGAUGCGGCUGAAGGAGGCACGGCUGCAUCCAUGUCCACGGAUGUGCCUGUGCUGUCGGAA
UGUCCGGGUCAAAGGCGUGGAUAGUCGGUGGCCGUAGGAUCCUUAUCGUCGGUGAAGCCUCCACGCC
GACGGUAAGGGUCCAAGCUCAGUGCAUUACGCUGUUACAUCGCACGCGCACCUCCAACUCCACC
```

Secondary structure:

```
.....................((((((..[[[[[[.))))))..((((((..[[[[[[.))))))..((
(((((((((((((..[[[[[[.)))))))))))))))).(((((((((((((((((((((((
(((((((...(((((((((((((((((((((((.(((((((((((((((..]]]]]]).))))))
))))))))))))..(((((((((((..[[[[[[.)))))))))))...((((((..[[[[[[.))
))))..)))))))))))))))))))).(((((((((((((((((((((..(((((((..]]]]]
].))))))).(((((((((((((((((..[[[[[[.))))))))))))))))).((((((((((((
(((((((((((((((((((...(((((((((((((((((((((.((((((((((((((((..]
]]]]].))))))))))))))))))..(((((((((((..]]]]]].)))))))))))...(((((
((..]]]]]].))))))).)))))))))))))))))))))..(((((((..]]]]]].))))))).
(((((((((((((((((..[[[[[[.))))))))))))))))).)))))))))))))))))))))
))))))))))...)))))))))))))))))))))).(((((((((((((((((..]]]]]].))))
))))))))))))..)))))))))))))))))))))))))))))))))).............
```

Table A.3: Triangulated prism.

A.1.4 Sterna

Source code for Sterna is available from `https://github.com/Ritkuli/sterna`.