# A Versatile Python Package for Simulating DNA Nanostructures with oxDNA

Kira Threlfall
*University of Arkansas, Fayetteville*

# A Versatile Python Package for Simulating DNA Nanostructures with oxDNA

An Undergraduate Honors College Thesis

in the

Department of Computer Science and Computer Engineering

College of Engineering

University of Arkansas

Fayetteville, AR

by

**Kira Threlfall**

# Abstract

The ability to synthesize custom DNA molecules has led to the feasibility of DNA nanotechnology. Synthesis is time-consuming and expensive, so simulations of proposed DNA designs are necessary. Open-source simulators, such as oxDNA, are available but often difficult to configure and interface with. Packages such as oxdna-tile-binding provide an interface for oxDNA which allows for the ability to create scripts that automate the configuration process. This project works to improve the scripts in oxdna-tile-binding to improve integration with job scheduling systems commonly used in high-performance computing environments, improve ease-of-use and consistency within the scripts composing oxdna-tile-binding, and move the oxdna-tile-binding source repository to GitHub for improved distribution.

# Acknowledgements

The author's work was supported in part by NSF grant CAREER-1553166.

# Contents

# List of Figures

## 1 Introduction

Recent developments in our understanding of DNA led to theorizing on how to use DNA molecules to perform computations or create structures at the nano-scale. DNA nanostructures have become a point of interest for nanotechnology since Seeman described the Holliday junction [19]. DNA structures are self-assembling, compact, and configurable, which makes them practical for a nano-scale material.

With the development of custom DNA synthesis, DNA nanotechnology has been shown to have practical use in medicine, and theoretical use in studies of computation and self-assembling systems. DNA synthesis is time consuming and expensive, so simulating designed structures to determine if they will behave properly is necessary before investing in synthesis. Thus development of these structures benifits from design, simulation, and analysis procedures before synthesizing the DNA in a laboratory setting.

Fast simulation of these structures is necessary to minimize research and development time and costs. Research groups have developed many simulators for this purpose, but they are relatively new. Because of the many features, configuring these tools, specifically oxDNA (an open source molecular dynamics simulator for DNA-based systems [26]), can take significant effort. This combines with the fact that research and development is an iterative process, causing researchers to spend large amounts of time on configuring and managing the simulations. These simulations can be performed on high-performance computing clusters. This requires the ability to schedule and organize jobs on the clusters' shared resources, such as graphics processing units (GPUs) or file I/O. The work for this thesis focuses on improving a Python package, oxdna-tile-binding developed by Kyle Sadler at the University of Arkansas, which includes a Python interface for oxDNA and several scripts to ease the simulation process. The improvements primarily concern the usability of oxdna-tile-binding specifically with a job scheduler.

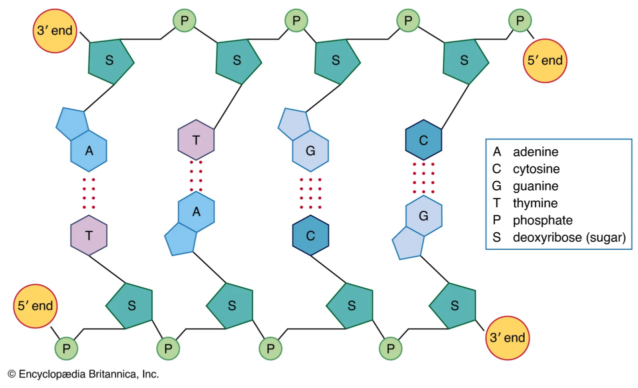
In Section 2 we will cover the structure of DNA in depth. This will allow us to understand the current standards for DNA structure construction and explore the many ways DNA technology can be made dynamic. We will then cover the process of DNA nanotechnology creation using available tools in Section 3. We will give a brief overview of many useful tools for design, simulation, and analysis, but we will focus on the simulation process using oxDNA. Section 4 will then cover how we have extended the oxDNA framework with a Python interface to allow for greater usability for oxDNA simulations. Finally, Section 5 will provide some examples of how to use scripts in this package and their results. We also provide, as an appendix, the README file which will be distributed with the python package. This file includes set up instructions and detailed descriptions on how to use the scripts in oxdna-tile-binding.

## 2 DNA Nanotechnology Overview

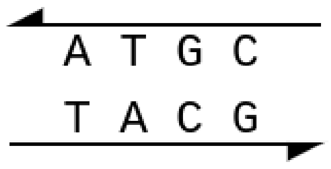
### 2.1 The structure of DNA

First we'll overview the structure of DNA. Two strands of DNA make the traditional double helix structure most are familiar with. Each of these strands are a chain of nucleotides. There are four distinct types of nucleotides that consist of four distinct nucleobases – adenine, guanine, cytosine, and thymine. We refer to these nucleotides as A, G, C, and T respectively. A and T nucleotides will only attach together and G and C nucleotides will only attach together due to the structure of the sugars [9]. When attached, we call these connections base pairs or nucleotide pairs. Also due to the structure of the nucleotides, DNA strands are directional. We refer to the ends of the DNA strand as the 3' ("three-prime") and 5' ("five-prime") ends. The traditional double helix consist of two anti-parallel DNA strands with complementary nucleotides (as shown in Figure 1a on the next page). Abstract diagrams of DNA are generally shown without the double helix, and the 3' end is generally denoted with an arrow (as shown in Figure 1b on the next page).

We consider the double helix to have made a full rotation when the direction of a single nucleotide has turned 360 degrees. The double helix makes a full rotation about every 10 to 10.5 base pairs — about 3.5nm [20]. The double helix form of DNA is also very rigid while the single stranded form is pliable. Because there are 4 different nucleotides and each strand is directional, there are $4^n$ unique strands of DNA that are of length $n$



(a) An example of the molecular structure of DNA from [9]



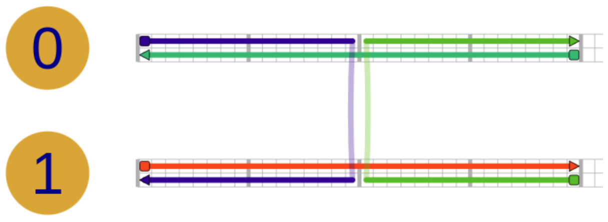
(b) An abstract visualization

Figure 1: Different visualizations of DNA

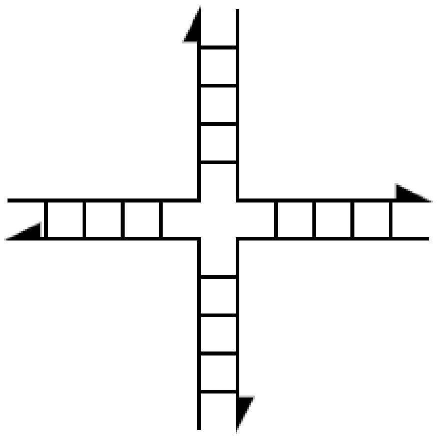
nucleotides. This combination of properties — small, rigid or pliable, and configurable — made DNA a prime candidate for a new technology.

## 2.2 The first steps in DNA nanotechnology

DNA nanotechnology began in the 1980's when Nadrian Seeman showed how to create a 2D structure called the Holliday junction with four strands of DNA [20] [19]. See Figure 2 for an example of this junction.



(a) Designed in scadnano [8]



(b) Adapted from [19]

Figure 2: Various visualizations of the Holliday junction

In order for the junction to be stable, the DNA strands must have unique base pair patterns. Without careful consideration, the junction might resolve itself into many different forms like the one in Figure 3. Rules to avoid this were presented by Seeman in

[19]. These rules mostly revolve around avoiding patterns.

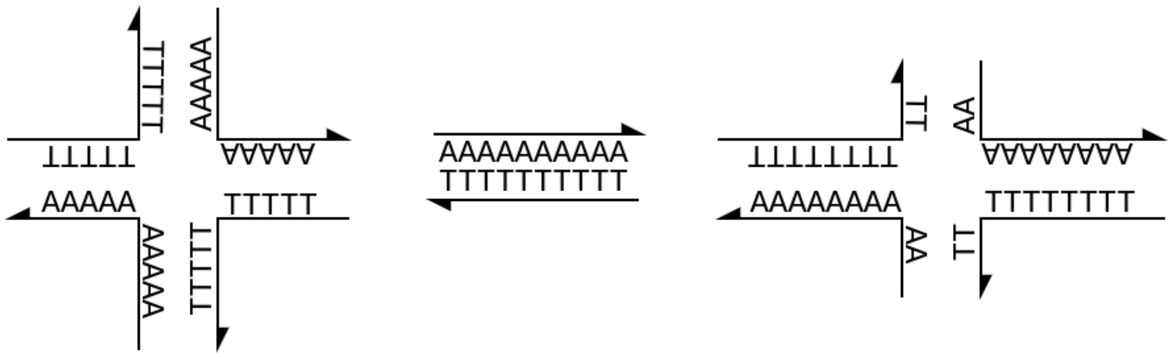


Figure 3: An example of two DNA strands that may take several forms

## 2.3 DNA origami

In 2006, Rothemund introduced the first concept of DNA origami [6]. Rothemund's origami used a long circular single strand of DNA as a “scaffold” and short single strands of DNA as “staples.” These staples, when constructed, cause the scaffold to pinch together. For example, consider a scaffold containing the sequence ATCGGGAT at one location and GGATCGTC at another location. Half of staple with the construction TAGCCCTA CCTAGCAG will attach to the first location and the second half will attach at the other location, pinching them together. See Figure 4 for a visualization of this.

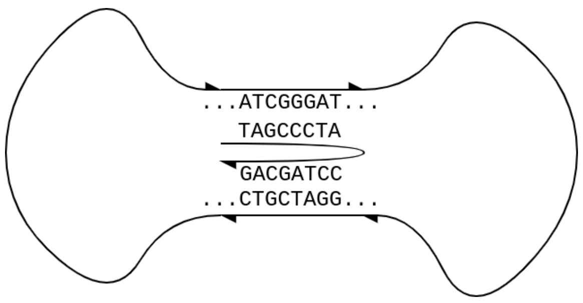


Figure 4: A visualization of a staple on a scaffold

Using this method, we can design a shape by laying out a scaffold and placing staples where the scaffold should be held together. The scaffold Rothemund used for his experiments was a naturally occurring circular single-stranded DNA from the M13mp18 virus which is 7,249 nucleotides (about 2415nm) long [17]. This DNA strand could also be cut for designs requiring a non-circular scaffold. He designed several 2D shapes (one of which is shown in Figure 5), synthesized the staples, then mixed the scaffolds and staples in a solution.

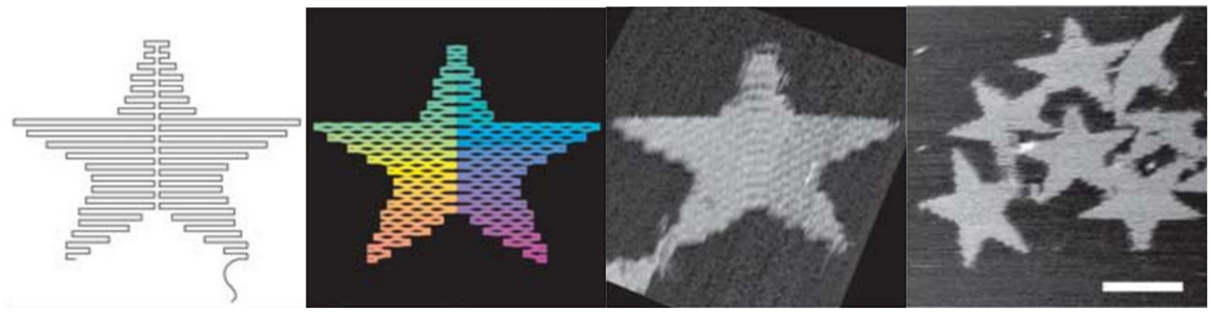


Figure 5: An example of a star made with a M13mp18 scaffold and staples from [17]. From left to right: the scaffold path, the scaffold with staples, and the atomic force microscopy (AFM) images of the DNA structure

We can also create 3D structures by “stacking” 2D slices created with a scaffold (similar to some forms of 3D printing) [4]. These are usually created on a square lattice or honeycomb lattice as shown in Figures 6 and 7. We call each point in the lattice a double-helical domain. 2D origami is designed on a square lattice that just extends across one row or column.

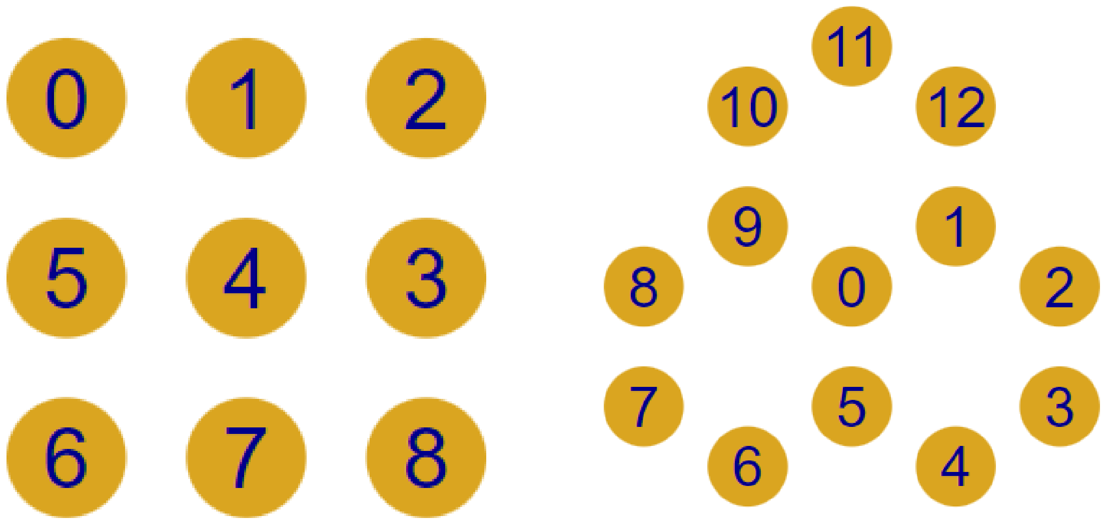


Figure 6: Example of a square (left) and honeycomb lattice (right) in scadnano [8]

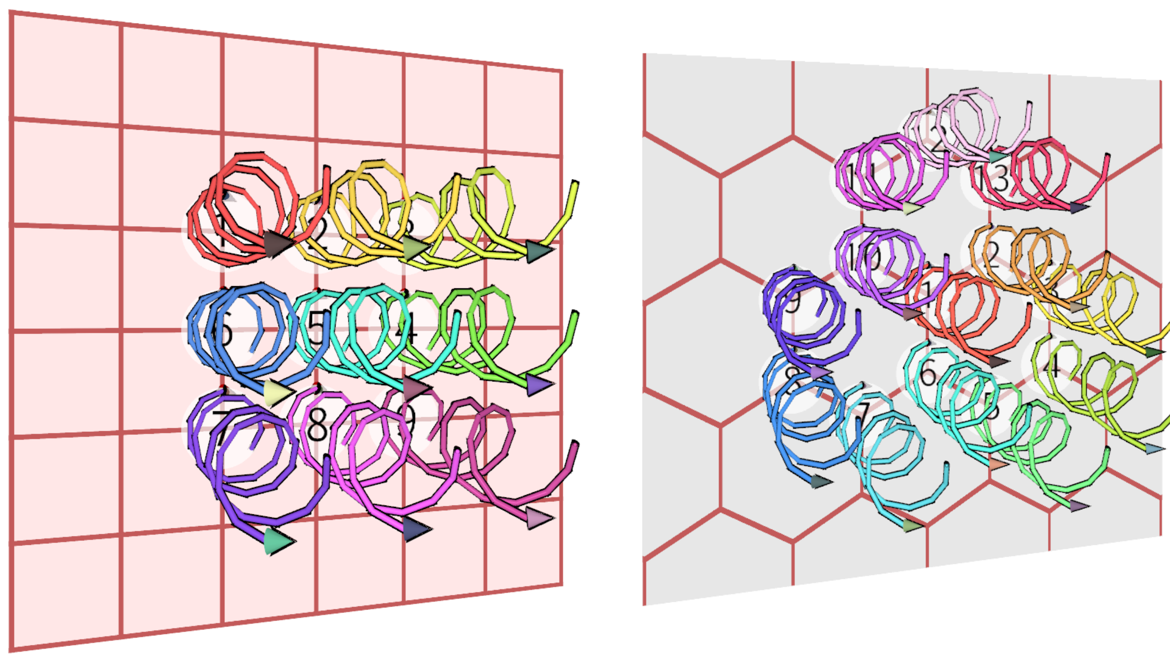


Figure 7: Example of a square (left) and honeycomb lattice (right) in ENSnano [11]

When designing DNA origami, we must consider the physical properties of the DNA. Where a staple attaches to two different locations on the scaffold is called a crossover, and where these crossovers should occur depends on which lattice structure is being used.

## 2.4 Designing on a lattice

Ideally, distance between crossovers between two domains would occur in a multiple of 10.5 base pairs that is divisible by the maximum number of neighbors possible for a single domain.

For the honeycomb lattice, each double-helical domain has up to three neighbors. 21 is a multiple of 10.5 and divisible by three, so crossovers between two domains should occur every 21 base pairs in a design. Since there are three neighbors to each double-helical domain, crossovers from each domain should occur every 7 base pairs [4].

For the square lattice, each double-helical domain has up to four neighbors. Unfortunately if we were to use an exact multiple of 10.5, it would require a distance of 84 base pairs (28nm) for crossovers between two domains. This is too far for small structures and may result in gaps between each double-helical domain. For a square lattice, it is recommended that crossovers from a helical domain occur about every 8 base pairs (and thus crossovers between two helical domains will occur about every 32 base pairs) [4]. This is because if we assume a double helix makes a full rotation every 10.67 base pairs (instead of 10.5), we can use a 32 base pair distance for crossovers between two domains because 32 is a multiple of 10.67 and divisible by four.

Maintaining exactly 8 base pairs between crossovers when using a square lattice can cause twisting torques because the assumption that the double helices make a full rotation every 10.67 base pairs is not accurate. To counteract this, it is recommended that designs alternate crossover distances between 7 and 8. Maintaining an average of 10.5 or 10.4 base pairs between crossovers from one double-helical domain to another is recommended for a square lattice. Deviating from the 7 base pair distance for a structure on a honeycomb lattice can also cause twisting of the structure [4].

## 2.5 Curved and off-lattice DNA designs

This twisting is most obvious on 2D designs since there are no forces from other helices to help counteract the twist.

We can also exploit this tension to create curved designs as shown by Dietz, Douglas, and Shih in [7]. They used a bundle of DNA arranged on a honeycomb lattice and varied the base pair distance between staples. Using exactly 7 base pairs per crossover resulted in little to no twisting, using 5 base pairs per crossover resulted in a left-handed torque, and 9 base pairs per crossover resulted in a right-handed torque. They also used insertions and deletions to induce a precise bend in the bundle. This resulted in the outer layer of the curved bundle having a greater number of base pairs than the inner layers. With these results, we can create non-linear designs such as a mesh sphere or gears [7].

It is also possible to make wireframe assemblies from DNA. In 1991 the first of these wireframe DNA structures was created (a cube), and recently, it has been shown that we can create arbitrary 3D meshes of shapes by triangulating those shape [21]. An example of a wireframe cube is shown in Figure 8. Polygon triangulation is a method used most commonly in computer graphics to draw arbitrary surfaces. These meshes can used as cages to transport other molecules for medicinal use [21]. These wireframe assemblies are considered to be off-lattice designs, but can still be designed using a lattice-based program.

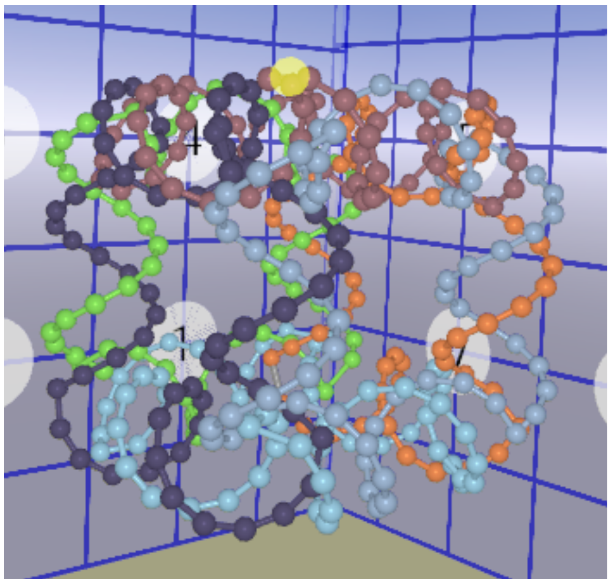


Figure 8: A cube designed in ENSnano [11] where a single DNA strand correlates to each face of the cube

## 2.6 Dynamic DNA

The nucleotides of DNA are not static, they experience Brownian motion like all small particles suspended in fluid. This motion makes dynamic DNA structures possible as random nucleotides pairs will disconnect at random.

We can use several different techniques to make dynamic DNA structures. Most of these techniques use some form of 'trigger' strand (or molecule) to cause a disruption in the preexisting structure. Many of these techniques take advantage of the fact that DNA strands will prefer to have more nucleotides attached to minimize the free energy of the system [21].

An example of dynamic DNA structures in a medical use has been shown by Li et al who have constructed autonomous DNA robots to deliver treatment directly to a tumor site [12]. Each nanorobot starts as a 90nm $\times$ 19nm $\times$ 2nm tube holding an active thrombin. The robots open when in contact with specific nucleolin proteins from the tumor as shown in Figure 9.

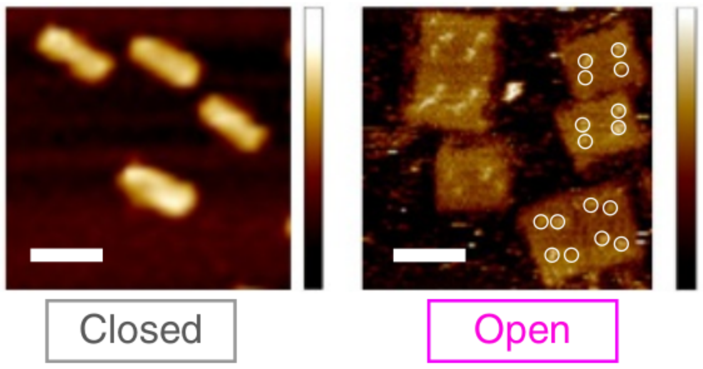


Figure 9: MFA imaging of the nanorobots from [12]

## 2.7 Self-Assembling tile models

Because there are many unique combinations of sequences for DNA strands, we can perform computations with DNA. The most popular systems of computation used in DNA are tile based, and the three we will briefly discuss here are Wang tiles, the aTAM, and the kTAM.

Wang tiles are square tiles with colors on each edge. These tiles attach whenever they have an edge that matches. It is easy to design Wang tiles as Holliday junctions where the arms of the junction have excess nucleotides as shown in Figure 10. The sequence of the excess nucleotides acts as the 'color' of each side of the tile. The self-assembling of these tiles can attach to make patterns (or perform computations) [20].

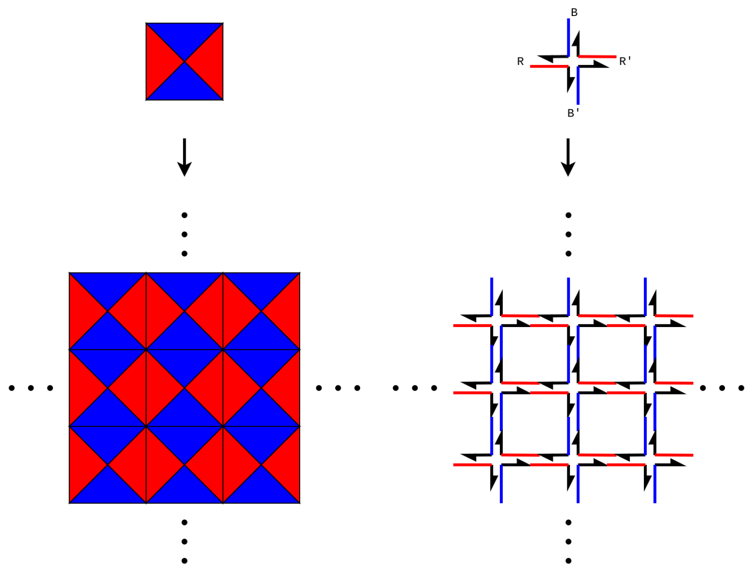


Figure 10: An example of a Wang tile made with a Holliday junction adapted from [20]

Notice, in the DNA implementation, the sequences (or colors) of the sides must have a corresponding complement for the sides to match. This does not interfere with the Wang tile model as Wang tiles are directional and cannot rotate or flip.

The abstract Tile Assembly Model (aTAM) was developed to add upon the Wang tile model. In aTAM, each side of a tile has a glue type and strength (in contrast to the Wang model where each side only has a type). Tiles attach if their sides are the same glue type and their strength is greater than or equal to the temperature (where strength and temperature are natural numbers). Tiles can also attach if the sum of the strength of the edges is greater than or equal to the temperature [14]. Each tile set starts with a seed tile and adds one tile at a time.

Wang tiles do not consider time and growth, and every edge must match the edge it is attached to. In the aTAM, tiles are allowed to have mismatched sides as long as the tile is attached with sufficient strength on other sides [14].

In the aTAM, tiles can self-assemble to perform computations or make finite shapes, such as a square, or infinite shapes, such as a binary counter (A binary counter is any structure that displays the sequence of incrementing binary numbers usually starting at 0 or 1). Both Wang tiling and the aTAM are Turing complete, but because the aTAM has a conception of time, it is closer to reality of how DNA attaches [14].

Similar to how Wang tiles can be produced in DNA, so can aTAM tiles. If we were to use Holliday junctions to create aTAM tiles, we would simply make the sequence for the glue type longer for larger strengths. The temperature in the aTAM tile set corresponds to the temperature of the solution the DNA tiles are in and the concentration of the tiles in the solution. The higher the temperature, more nucleotides need to attach for the tile to remain attached.

The kinetic Tile Assembly Model (kTAM) attempts to better model how DNA tiles realistically attach. In the kTAM, tiles are allowed to attach similarly to aTAM. A tile may also attach temporarily even if it does not have a sufficient strength to remain attached, and tiles may detach[14]. Due to this change from aTAM, there are three general types of errors that can occur in the kTAM.

The errors in the kTAM are as follows [14]

1. Growth errors: a tile temporarily attaches with **missmatched glues**, but before it can detach, another tile bonds to it with the strength needed to keep the error tile in place.

2. Facet errors: a tile temporarily attaches with **insufficient strength**, but before it can detach, another tile bonds to it with the strength needed to keep the error tile in place.

3. Nucleation errors: when tiles attach to each other to create a another "seed" assembly.

Since the kTAM more closely represents how DNA tiles attach, if we were to use tiles to compute, we should expect the errors that occur in the kTAM. Therefore we should create tile sets that can reduce errors. There are many ways to do this as discussed in [14].

If we want to create a physical tile set and test it in a lab, we would want have some expectation of its performance. Synthesizing the DNA is expensive and time consuming. Simulation is a way to gain confidence in a DNA structure design

# 3 Tools for DNA Nanotechnology Creation

These DNA structures are difficult to design by hand, so tools to aid in the design have been developed. When designing DNA nanotechnology, we want to have a high chance of success before ordering the physical DNA. To do this, we first design the DNA nanotechnology with a design tool, simulate it, and analyze how it performed. We will cover three designing tools, three simulation tools, and two viewing tools.

## 3.1 DNA design tools

Wyss Institute for Biologically Inspired Engineering at Harvard University maintains cadnano [5], the most popular tool for DNA structure design, and one of the most supported.

At a high level, it is a program to design these DNA structures on a lattice in preparation for simulation or real-world experiment. It can be used to design tiles for use in a tile assembly, or design DNA origami/nanotechnology [6]. Users may use a honeycomb or square lattice depending on their design goals. For creating a DNA origami with a scaffold, users would place the scaffold in the desired shape then place staples on the scaffold to create the connections. cadnano offers a 3D view (with support of additional software) of the structure being designed, but no 3D interactivity [5][6]. The files that cadnano exports have become commonly supported by other programs (for design, simulation, and viewing).

A similar tool is scadnano [8] ("scriptable-cadnano") which was built to mimic the appearance of cadnano with the additional features to accelerate design of DNA nanostructures. Its primary additions over cadnano are a web-based interface, tight Python integration for scripting, and a file type that is easily human readable. This program has the same disadvantage as cadnano as it focuses its support on lattice-based DNA structure construction.

There is an experimental autostaple feature in scadnano which attempts to alleviate some of the tedious work of adding staples (shown in Figure 11).

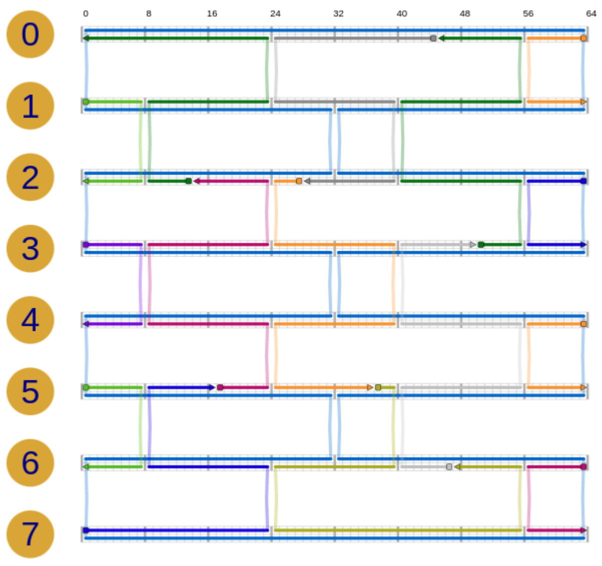
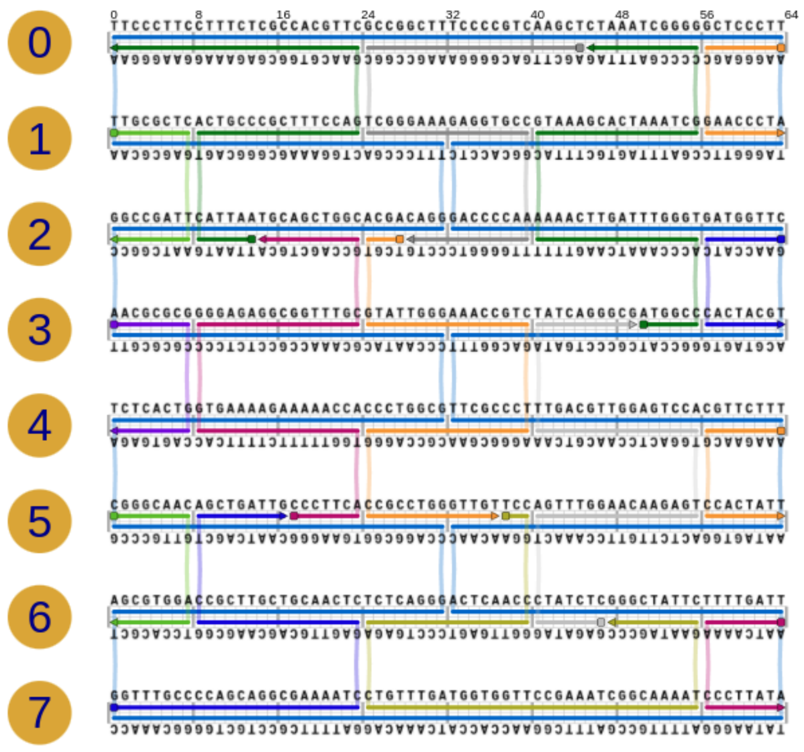


Figure 11: An example of using the scadnano auto staple feature on a rectangular scaffold [8]

Both cadnano and scadnano allow the user to assign DNA sequences to the scaffold and complementary sequences to the staples as shown in Figure 12 [8]. This allows users to assign sequence of the scaffold to be the sequence of a DNA strand that they have easily available (such as the M13mp18 DNA strand) for when they want to run a lab experiment.



0
8
16
24
32
40
48
56
64
0
TTCCCTTCCTTTCTCGCCACGTTCGCCGGCTTTCCCCGTCAAGCTCTAAATCGGGGGCTCCCTT
1
TTGCGCTCACTGCCCGCTTTCCAGTCGGGAAAGAGGTGCCGTAAAGCACTAAATCGGAACCCTA
2
GGCCGATTCATTAATGCAGCTGGCACGACAGGGACCCCAAAAAACTTGATTTGGGTGATGGTTC
3
AACGCGCGGGGAGAGGCGGTTTGCGTATTGGGAAACCGTCTATCAGGGCGATGGCCCACTACGT
4
TCTCACTGGTGAAAAGAAAAACCACCCTGGCGTTCGCCCTTTGACGTTGGAGTCCACGTTCTTT
5
CGGGCAACAGCTGATTGCCCTTCACCGCCTGGGTTGTTCCAGTTTGGAACAAGAGTCCACTATT
6
AGCGTGGACCGCTTGCTGCAACTCTCTCAGGGACTCAACCCTATCTCGGGCTATTCTTTTGATT
7
GGTTTGCCCCAGCAGGCGAAAATCCTGTTTGATGGTGGTTCCGAAATCGGCAAAATCCCTTATA

Figure 12: The DNA rectangle in Figure 11 assigned with the predefined M13mp18 sequence provided by scadnano [8]

ENSnano expands upon cadnano and scadnano by adding user experience features. It allows users to import cadnano and scadnano designs, recommend 3D crossovers, and offer 3D visualization and editing [11]. The 3D editing allows users to easily design structures without a lattice-bound scaffold such as a wireframe polyhedra as shown previously in Figure 8.

## 3.2 Simulating

Once these DNA designs have been created, they are simulated before any lab experimentation to verify fitness. Simulation can help detect errors in a tile system, twisting torques in DNA origami, and confirm functionality of DNA robots.

MrDNA is an open source Python package for simulation. It is a more recent simulation framework that promises multi resolution simulations in under thirty minutes while offering better support for off-lattice structures [13]. It requires an Nvidia GPU in a super computing cluster to use, and it is not as well supported as it is one of the newest simulation tools[6].

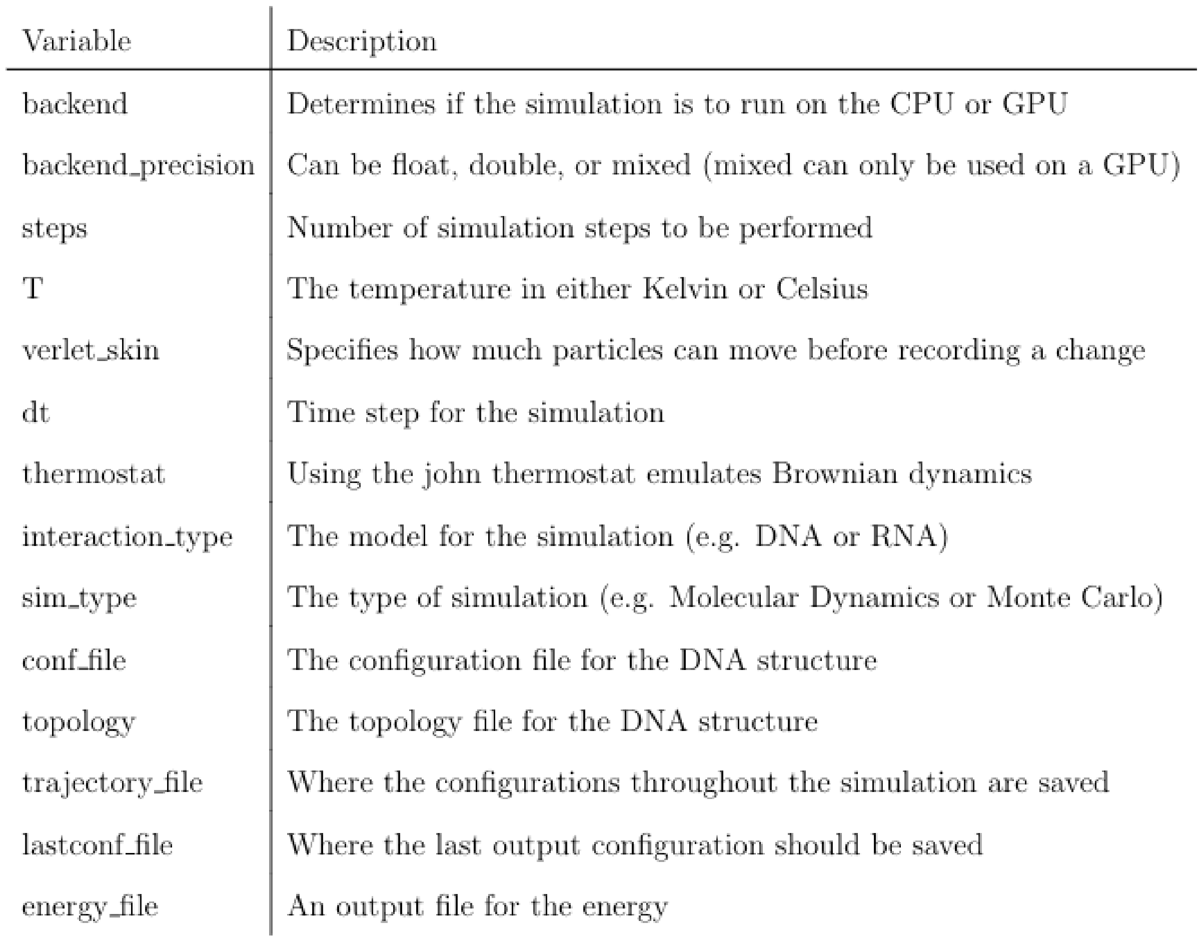
CanDo takes cadnano formatted design files and performs some simulations to transform lattice bound designs into 3D shapes. It also provides the user with a heatmap of how flexible each point in the structure is [4]. This heatmap is useful for determining if a DNA design needs any more revisions. It also now offers the ability to generate an atomic model of the DNA structure, and has the option to include the DNA sequences if provided by the user in a sequence file. CanDo's primary use is for iterative designing of origami that may have curves, twists, or an off-lattice design [4]. CanDo can be accessed here [22] for free.

MrDNA and CanDo are newer simulation tools, and have shown promising results. The most commonly used tool for DNA simulation, and the one we used in this paper, is oxDNA.

## 3.3 Performing and analyzing simulations using oxDNA

The most supported simulation tool is oxDNA which uses Monte Carlo and Molecular Dynamics to simulate interactions between nucleotides [26]. The file types used for oxDNA have become somewhat standard as all three of the aforementioned design tools in Section 3.1 allow the user to export the design to oxDNA for simulation. oxDNA has many parameters to control factors of the simulation including temperature, properties of the simulated solution the DNA molecules are suspended in, and external forces. It also allows for parameters in how the simulation is run such as which type of molecular dynamics to simulate with, backend precision, and whether to run with the CPU or an Nvidia GPU [26].

oxDNA is open source and can be downloaded for free. Setting up oxDNA is simple, it uses cmake to configure the build system and then GNU make to compile oxDNA [26]. To use oxDNA, we run the command `oxDNA input_config.dat` with an input configuration file that contains the variables necessary for oxDNA to run. Below in Figure 13 are some of the necessary variables for an input file. These variables control general, simulation specific, and input/output options.



| Variable | Description |
|---|---|
| backend | Determines if the simulation is to run on the CPU or GPU |
| backend_precision | Can be float, double, or mixed (mixed can only be used on a GPU) |
| steps | Number of simulation steps to be performed |
| T | The temperature in either Kelvin or Celsius |
| verlet_skin | Specifies how much particles can move before recording a change |
| dt | Time step for the simulation |
| thermostat | Using the john thermostat emulates Brownian dynamics |
| interaction_type | The model for the simulation (e.g. DNA or RNA) |
| sim_type | The type of simulation (e.g. Molecular Dynamics or Monte Carlo) |
| conf_file | The configuration file for the DNA structure |
| topology | The topology file for the DNA structure |
| trajectory_file | Where the configurations throughout the simulation are saved |
| lastconf_file | Where the last output configuration should be saved |
| energy_file | An output file for the energy |

Figure 13: Some of the necessary variables for an oxDNA input file [26]

An example of an oxDNA input file is shown in Figure 14 on the next page.

Some oxDNA files when exported from cadnano orient the design in a planar fashion. This is reasonable for most 2D designs, but some 3D designs have staples that go across multiple rows when designed flat. When exported like this, it may be difficult to simulate using oxDNA's molecular dynamics. Relaxation decreases the distance between nucleotide pairs, and makes it possible for simulation with oxDNA. To relax using oxDNA, we can change the parameters to use more course simulation.

```
backend=CPU
backend_precision=double
dt=0.005
interaction_type=DNA
max_backbone_force=5.
no_stdout_energy=false
time_scale=linear
print_energy_every=2000
refresh_vel=1
sim_type=MD
steps=2000
newtonian_steps=103
diff_coeff=2.50
thermostat=john
T=45C
verlet_skin=0.05
rcut=2.0
restart_step_counter=0
print_conf_interval=100
print_conf_ppc=51
external_forces=0
max_io=20
log_file=./log.dat
energy_file=./energy.dat
trajectory_file=./trajectory.dat
topology=rect.top
conf_file=rect-relax_test.conf
lastconf_file=rect-simulate.conf
```

Figure 14: An example of an input file for oxDNA

If we relax a structure and then simulate it, this will take at least two separate oxDNA calls with extremely different input files. Relaxation itself may take more than 1 simulation to achieve an appropriate configuration file for molecular dynamics simulation.

Creating these input files and running multiple simulations to achieve one final result can be inconvenient and time consuming. To make this simulation process more convenient, we have created a Python package to interface with oxDNA which is discussed in Sections 4 and 5.

## 3.4 Viewing DNA

After simulation, we need to view the resulting structure to see how well it fared. For example, we may want to check that several DNA tiles are realistically compatible or that a DNA origami doesn't warp because of tension. To show how the two following visualization and analysis tools differ, we will be using the example shown in Figure 15.

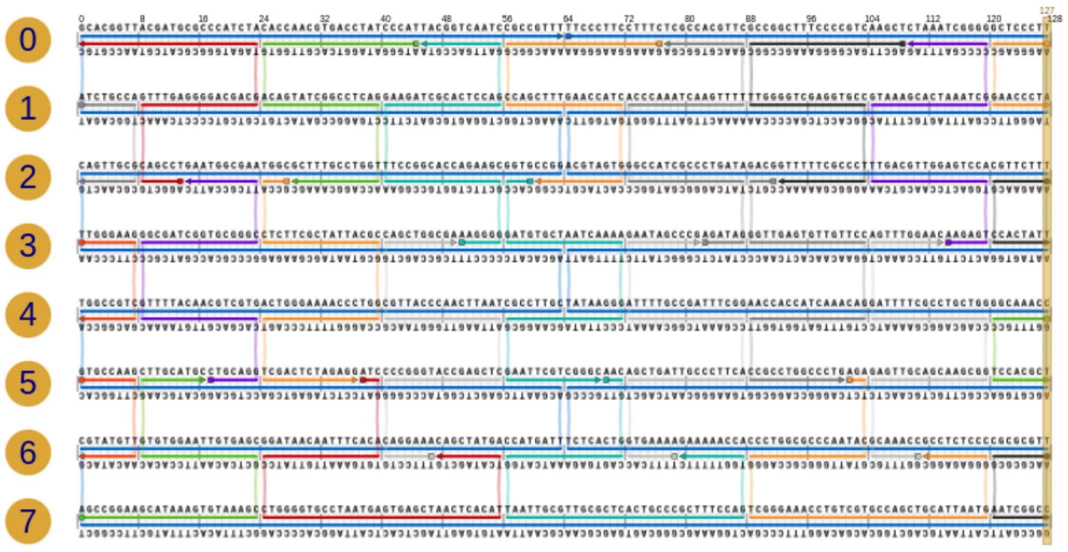


Figure 15: A rectangle designed in scadnano using the auto stapling feature [8]

oxView is a web browser-based visualizer for DNA designs. To view results from a simulation, users can simply drag and drop oxDNA trajectory and topology files into the web application. It also allows for basic edits to DNA designs and some relaxation capabilities [15]. Relaxation in oxView uses rudimentary rigid body simulations to rearrange the design to a usable form for oxDNA. An example of viewing a simulated DNA structure is shown in Figure 16 on the next page.

Viewing with oxView only allows for nucleotides to be viewed as rigid bodies. For more detail, Chimera and ChimeraX are available. Chimera is used to visualize and analyze molecular structures [24]. While oxView may be useful for quick visualization and adjusting designs, Chimera is useful for detailed visualization and gathering final imaging. Chimera allows for several different viewing options and the ability to render images with specific lighting. ChimeraX has also been recently developed which offers better

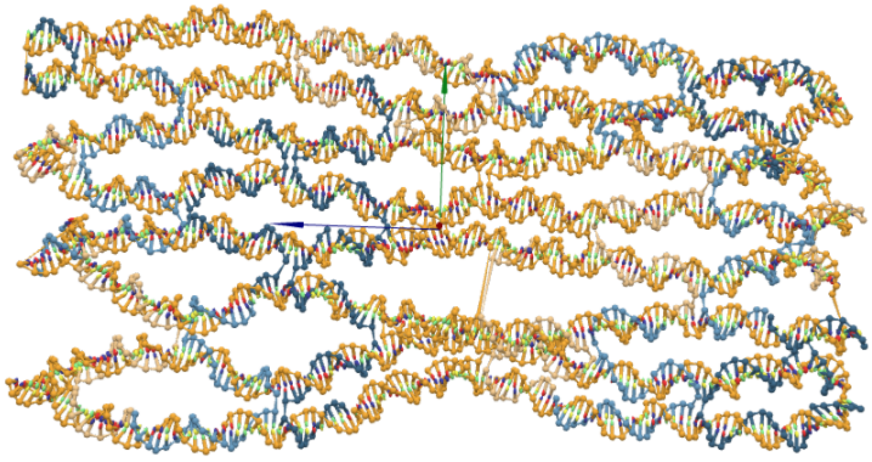


Figure 16: The DNA rectangle from Figure 15 after being relaxed and simulated with oxDNA using the input file in Figure 14 as viewed in oxView [15]

performance and some new features (but is also missing some features from Chimera) [24]. An example of the same simulated design from Figure 16 is shown in Chimera below in Figure 17.

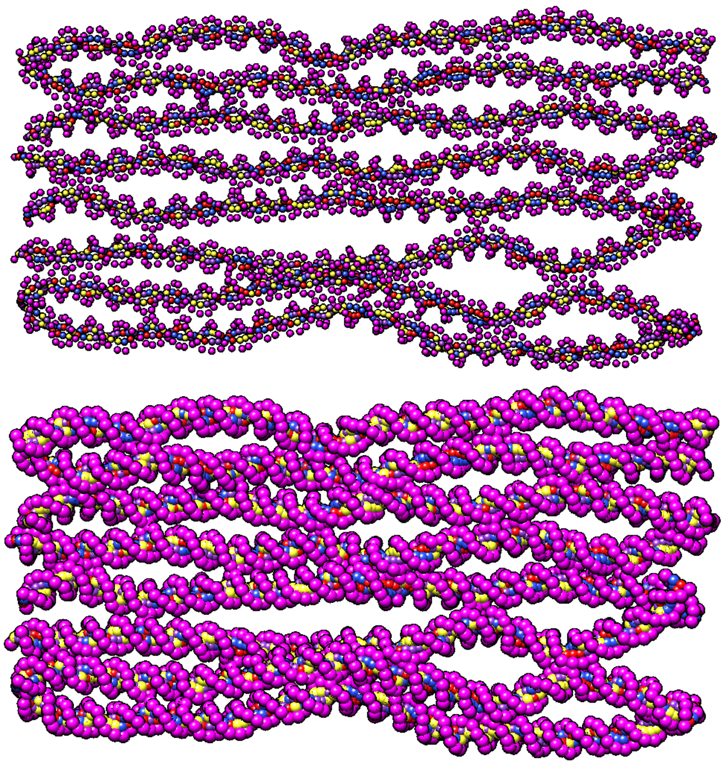


Figure 17: The simulated DNA rectangle from Figure 15 as viewed with Chimera using stick (left) and ball and stick (right) models [24]

Notice below in Figure 18 that the center is so warped that a staple connection is incredibly stretched out. Also notice that the ends are twisted. This structure may not form completely in a real experiment because it requires the DNA strands to twist in unnatural ways. This DNA structure would need revisions before being used. Once revisions are made, we should simulate it again to check for improvements or any other problem areas. We may also want to simulate several times to see if this structure has different random outcomes.

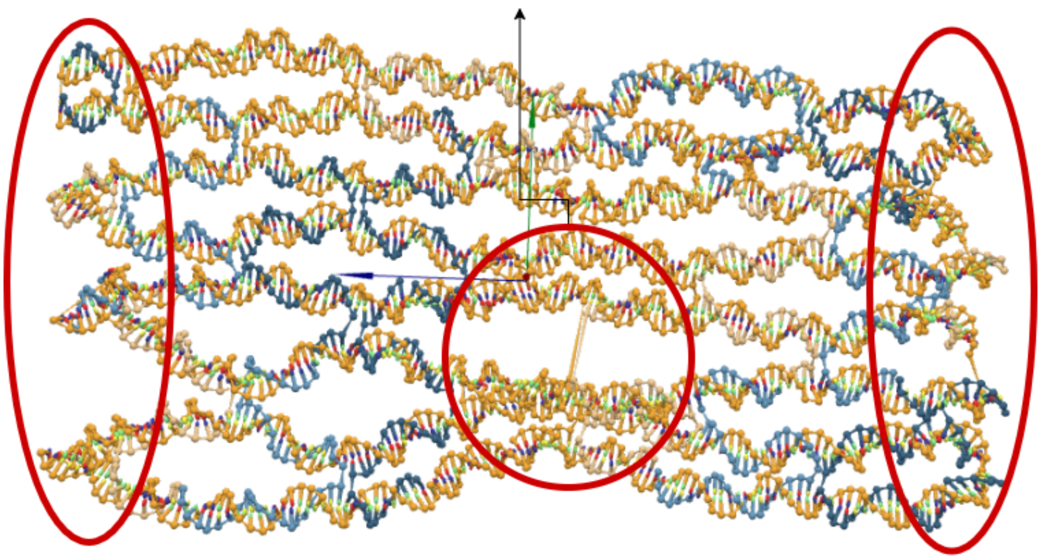


Figure 18: Problem areas in a DNA rectangle

If we were to be looking at a tile assembly, we may want to check if two tiles we design will connect or not. We can't just trust one simulation, and looking at it visually may be too time consuming. It would be beneficial to be able to generate scripts that allow for repeated simulation and analysis automatically.

## 4 A Python Package for oxDNA Simulations

We created a Python package, oxdna-tile-binding, that assists in the simulation of DNA structures using oxDNA, specifically on high performance computing clusters. It contains another Python package (pyoxdna) and some useful scripts to help setup, general simulation, and tile binding analysis. Most of the scripts in oxdna-tile-binding, such as tile_binding_auto_monitoring_light.py (Section 7.3.8), were created to support the development of DNA tiles for computations as discussed in section 2.7 by allowing easy simulation of these tiles. By using this Python package to assist with simulations, we can decrease the development time of these DNA structures.

## 4.1 pyoxdna

pyoxdna is a Python interface for oxDNA. It contains the DNARelaxer class, a utils module, and the pyoxdna class.

DNARelaxer is a class for automatic relaxation. It uses oxDNA to relax the DNA structure until it passes a test simulation or achieves a maximum number of attempts.

The utils module offers the main tools that pyoxdna uses, and allows for users to access those tools. It includes functions to modify directories and files, read and write configuration files, and get a standard set of configuration files stored in pyoxdna/configs. It also contains a Metrics class that can help with the analysis of DNA structures with their configuration files.

The pyoxdna class is the main module for running oxDNA simulations. It manages outputs, files, and processes to help with writing scripts.

## 4.2 Moving Support for oxdna-tile-binding

Previously, oxdna-tile-binding was built to run on the Razor system offered by the Arkansas High Performance Computing Center (AHPCC) [25]. After Razor's deprecation, we had to move support for these scripts to another super computing cluster offered by the AHPCC. The scripts in oxdna-tile-binding have the ability to submit jobs on behalf of the user through a program called job_launcher.py. When oxdna-tile-binding was on Razor, to use any compute nodes, its programs required PBS (Portable Batch System [1]) scripts. These scripts specified the resources the job would need, the queue the job should go in, and various other parameters specific to the Razor system. The AHPCC offers two super computers, Trestles and Pinnacle. When choosing which cluster to move support to, we considered how easy it would be to move oxdna-tile-binding and how useful each computing cluster would be.

Trestles uses PBS [1], like Razor did. To move support to this system, We would only have needed to adjust the system specific parameters in job_launcher.py (such as the queue names). Trestles works best with small programs that only need a single node. There are about 200 public nodes, each having 32 cores and 64GB of memory.

Pinnacle uses Slurm Workload Manager [18]. The Pinnacle cluster offers about 100 compute nodes for various use cases. There are standard public compute nodes, public high-memory compute nodes, public GPU compute nodes, and some private nodes for varying groups at the University of Arkansas. The standard public and GPU compute nodes have 32 cores and 192GB of memory each with the addition of an NVidia V100 Tesla GPU for the GPU nodes.

One of the programs offered by oxdna-tile-binding is a program to test how long a user's system takes to run simulations with a varying amount of nucleotides using oxDNA with the CPU or GPU, so we had to move support for oxdna-tile-binding to Pinnacle to support this script.

When oxdna-tile-binding was formatted for Razor, the file job_launcher.py took parameters and created a PBS script to launch a job. When we first moved this to Pinnacle, we translated the PBS script creation to a Slurm script using resources found here [18]. When we decided to host oxdna-tile-binding on GitHub, we wanted to make it easy to run jobs on any system with any resource manager.

To do this, we edited the scripts to allow the user to input a job script. The input file should have the command the script should be run with in the first line and the script to be run in proceeding lines with a flag for the job name and the output file. An example of a job script is shown in Figure 19. The job launcher takes all but the first line of the file, adds lines to load modules, time the command, and initiate environment variables. An example of the resulting file is shown in Figure 20.

```
sbatch
#!/bin/bash
#SBATCH --job-name=[job_name]
#SBATCH -p gpu72
#SBATCH --nodes=1
#SBATCH --ntasks=6
#SBATCH -t 72:00:00
#SBATCH -o [out_file]
cd "/scratch/$SLURM_JOB_ID"
```

Figure 19: An example of an input job script for Slurm

```
#!/bin/bash
#SBATCH --job-name=example1
#SBATCH -p gpu72
#SBATCH --nodes=1
#SBATCH --ntasks=6
#SBATCH -t 72:00:00
#SBATCH -o /home/user/example1/example.out
cd "/scratch/$SLURM_JOB_ID"
module purge
module load mkl/18.0.2
module list
export HOME=/home/user
export PATH=$PATH:/home/user/oxDNA/build/bin
time python3 /scrfs/storage/user/home/example.py
```

Figure 20: An example of the resulting job.txt generated by job_launcher.py from the above input that will be run with sbatch. This example has been reduced for brevity.

Users also have the option to run programs locally by not including a job script whenever they run a program. By allowing users to provide a job script, we allow them to use resource managers that we might not account for. This also allows for users to have specific job scripts per task. For example, one simulation may need to use a different queue or partition of the cluster than another simulation to access the GPU or have a longer maximum run time. Below in Figure 21 is an example of parameters for another job script we might use for a task needing more nodes, less time, on a different partition of the cluster, and sending an notification email when the job has completed.

More about submitting scripts as jobs can be found in Section 7.3.2.

```
#!/bin/bash
#SBATCH --job-name=example2
#SBATCH -p cpu01
#SBATCH --nodes=4
#SBATCH --ntasks=2
#SBATCH -t 1:00:00
#SBATCH --mail-type=END
#SBATCH --mail-user=user@uark.edu
#SBATCH -o /home/user/example2/example.out
...
```

Figure 21: An example of a job script for a job needing more nodes, less time, and a different partition of the cluster

## 4.3 Moving to GitHub

The code for oxdna-tile-binding was previously held in a Subversion (SVN) repository [3]. Git (another version control software [23]) has eclipsed SVN's popularity, and is now more widely used. We wanted this code to be easily accessed by the research community, so we chose to additionally host it in a Git repository on GitHub [10].

Migrating from a SVN repository to a Git repository while keeping the version history is sometimes possible, but it is difficult. Since the SVN repository is still in use, we decided to only migrate the necessary files from the SVN repository to the new Git repository on the uark-self-assembly GitHub. This also gave us the chance to prune any deprecated programs and files to improve the user experience and usability.

For ease of access, we also created a comprehensive document for oxdna-tile-binding written in markdown so it could be saved as a README file on the GitHub. Markdown files are also easy to read even if a markdown viewer is not available, and they are standard on GitHub.

## 4.4 Documentation and Bug Fixes

This documentation explains how users should set up their environment, including how to set up oxDNA, the use of each program, and how to run it. It also includes common errors or problems users might encounter and how to fix them. While writing this documentation, any bugs encountered were fixed. The entire README file can be found in the Appendix (Section 7).

Many changes we made were focused on improving consistency between the programs and ensuring that the documentation was consistent. For example, some programs required the environment variable OXDNA_HOME to be pointed at the oxDNA folder, and some required it to point to oxDNA/build. To make this consistent, we changed all programs to assume OXDNA_HOME was pointed at the oxDNA folder. We also ensured that the setup instructions for oxDNA in the documentation included creating the build folder and running cmake in that folder.

There are also instructions in the documentation on how to set up the pyoxdna conda environment [2]. This environment helps when submitting jobs, so that programs know the location of important folders (such as OXDNA_HOME). Sometimes when running programs locally, even in the pyoxdna conda environment, the environment variables would not be set properly. To inform users of this common error, instructions were added on how to recognize when this is happening and how to fix it by setting the environment variables either temporarily or permanently.

## 4.5 Improving computation_experiment.py

Timing is an important aspect of these simulations. The faster the simulation, the quicker the process of designing is. We were curious as to how the timing differs when using oxDNA with the CPU vs GPU. Generally, using oxDNA with the GPU is faster for larger cases. Knowing the exact sizes of structures that are faster with the CPU or GPU can help decrease simulation time. We improved upon the existing computation_experiment.py by attempting to make the timing more accurate.

The previous timing was exclusively using perf_counter(). This function returns the clock time, which is okay to use as long as the program starts immediately and does not pause [16]. Because we are testing on a cluster, this is not always possible. The resource manager may wait to start a job or pause a job if there are not enough resources to support it or too many jobs running. Computation_experiment.py was changed to use both perf_counter() and process_time() offered by Python's time library. The Python function process_time() returns the current process time, which does not include when the process has been asleep [16].

To further add to precision, we included the Linux time command before running the script. Users can look near the end of the job.out file for their job to see the real, user, and system time offered by this command. This time unfortunately includes relaxation, which we may not want to consider.

Relaxation would often fail repeatedly while using the GPU and thus take significantly longer, so for this experiment, to test the simulation time, all simulations relax with the CPU regardless of whether or not they were to be simulated with the GPU or CPU. An example of running computation_experiment.py can be seen in Section 5.2.

## 5 Examples of using oxdna-tile-binding scripts

### 5.1 Setup

To use the scripts in oxdna-tile-binding, users must set up an Anaconda3 environment [2]. This is easily done by setting the necessary environment variables in config.yml as shown in Figure 22 on the next page, and activating the pyoxdna conda environment.

Further explanation of setup can be found in Section 7.1.

Following are some examples of using scripts in the oxdna-tile-binding package. Exact instructions to run these scripts can be found in Section 7.3.

### 5.2 computation_experiment.py

This script is used for testing the performance of simulations with oxDNA using the GPU or CPU. It generates a spiral structure with $2^n$ nucleotides for an input $n$. This structure is then relaxed to allow for oxDNA simulation, and then the simulation is timed. The simulation parameters are consistent between all simulations except for the size of the structure and if the simulation is run using the GPU or the CPU.

This script is important when deciding how users should simulate their structures.

```
# your home dir
HOME: '/home/user'

# the parent dir where you want to store simulation output.
# Each simulation creates a working directory in SIM_HOME
SIM_HOME: '/home/user/simulations'

# where oxDNA is stored
OXDNA_HOME: '/home/user/oxDNA'

# where pyoxdna is stored
PYOXDNA_HOME: '/home/user/oxdna-tile-binding/pyoxdna'

# email address for status updates
EMAIL_ADDRESS: 'user@email.com'

# set level for output of debug messages, 0 (least) to 5 (most)
DEBUG_LEVEL: 5
```

Figure 22: An example of the config.yml file for oxdna-tile-binding

The results of average perf_time() and average process_time per nucleotide are given in a result.csv file which can be evaluated using a program such as Excel. Users can also go into the job.out file in the simulation folders to retrieve the real, user, and system time given by the linux time command.

Ideally, users will run this script for $n = 1$ to at most $n = 17$ to determine what size of simulation is best supported by their CPU vs their GPU.

This script can be run two ways. If using a resource manager, users can run

```
python3 computation_experiment.py [start n] [end n]
[0 for CPU 1 for GPU] [job_file]
```

To run an analysis for a single simulation locally, users can run
`python3 run computation_experiment.py [n] [0 for CPU 1 for GPU].`

When using computation_experiment.py to profile a system, users should run multiple iterations of each size on both the GPU and CPU to properly asses their system.

Below in Figure 23 are the results from running computation_experiment.py on the GPU with the following command once

```
python3 computation_experiment.py 1 8 1 example.job
```

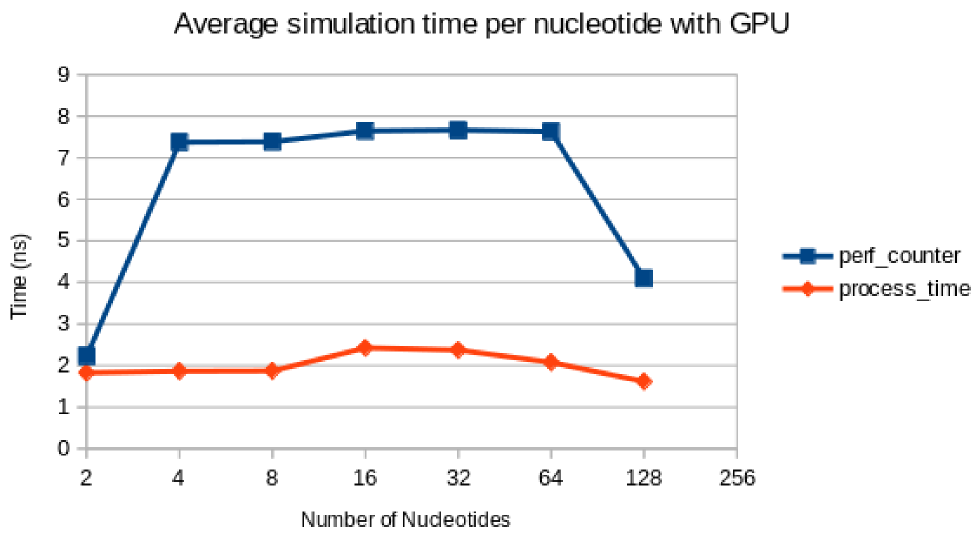


Figure 23: Results from computation_experiment.py

Notice that the average perf_counter results vary widely between simulations. This is because these simulations were run with many other nodes being occupied, so some simulations had to sleep for some time.

## 5.3 run_simulation.py

This script is an all-purpose simulation script. It takes a topology (.top or .dat) file and a configuration (.conf) file and produces a configuration file after relaxing and simulating for the specified number of steps.

Users can control if the simulation runs locally or with a job manager, if the DNA structure should be relaxed, simulated, or both, where to output the resulting files, how many GPUs to use, and more. Instructions on how to run this script can be accessed with `python3 run_simulation.py -h`.

Figure 24 on the next page shows a design was simulated using the following command

```
python3 run_simulation.py -j example.job -c square.dat
-t square.top -n 2000 -r -m
```

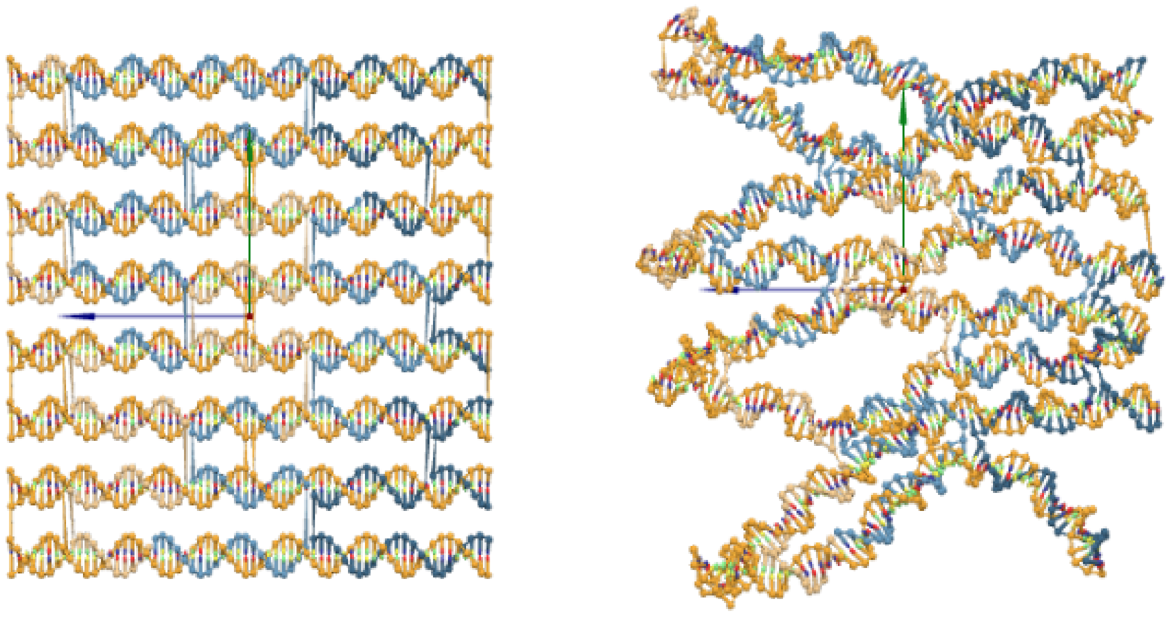


Figure 24: Auto stapled design in Figure 12 before simulation (left) and after (right) displayed in oxView [15]

The main benefit of this script is that it automates the creation of the input file for oxDNA according to parameters set by the user. It also automatically relaxes the DNA structure repeatedly until a test simulation passes. Simulation occurs immediately after relaxation. A downside of using this script is that if users would like to change any specific parameters that run_simulation does not allow input for (like simulation temperature), they would have to edit the simulation configuration files (which are stored in pyoxdna/configs).

## 6 Conclusion

This paper has described the improvements made to oxdna-tile-binding, and given an overview of the background literature for DNA nanotechnology. The improvements to usability of oxdna-tile-binding are expected to assist researchers in the development of new DNA structures. We discussed the structure of DNA, how that structure enables the use of DNA in designing nano-scale structures, the suite of design tools that enable researchers to build experiments for simulation, the improvements to oxdna-tile-binding undertaken as part of this research project, and examples of how to use some of the scripts scripts available in oxdna-tile-binding.

The following are improvements and opportunities for future work on the oxdna-tile-binding package:

1. Adding more customized scripts using the pyoxdna package
2. Further improving the interface consistency between existing scripts
3. General refactoring of the code base to ensure best separation of concerns between modules
4. Further utilizing oxdna's utilities to automate conversion to preferred filetypes for use in various visualization tools

Pursuing these improvements would further enable researchers to focus on the design and simulation of these structures without unnecessary concern for the implementation details of oxDNA.

# References

[1] "OpenPBS," Altair Engineering, Inc. [Online]. Available: https://www.openpbs.org

[2] "Conda," Anaconda, Inc. [Online]. Available: https://docs.conda.io/projects/conda/en/latest/index.html

[3] "Apache Subversion," The Apache Software Foundation. [Online]. Available: https://subversion.apache.org/

[4] C. E. Castro, F. Kilchherr, D.-N. Kim, E. L. Shiao, T. Wauer, P. Wortmann, M. Bathe, and H. Dietz, "A primer to scaffolded DNA origami," *Nature methods*, vol. 8, no. 3, pp. 221–229, 2011.

[5] N. Conway and S. Douglas, "cadnano." [Online]. Available: https://cadnano.org/index.html

[6] S. Dey, C. Fan, K. V. Gothelf, J. Li, C. Lin, L. Liu, N. Liu, M. A. Nijenhuis, B. Saccà, F. C. Simmel *et al.*, "DNA origami," *Nature Reviews Methods Primers*, vol. 1, no. 1, pp. 1–24, 2021.

[7] H. Dietz, S. M. Douglas, and W. M. Shih, "Folding DNA into Twisted and Curved Nanoscale Shapes," *Science*, vol. 325, no. 5941, pp. 725–730, 2009. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1174251

[8] D. Doty, B. L. Lee, and T. 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*, ser. Leibniz International Proceedings in Informatics (LIPIcs), C. Geary and M. J. Patitz, Eds., vol. 174. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 9:1–9:17. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/12962

[9] "DNA," Encyclopaedia Britannica, Mar 2022. [Online]. Available: https://www.britannica.com/science/DNA

[10] “GitHub,” GitHub. [Online]. Available: https://github.com/

[11] N. Levy and N. Schabanel. ENSnano. [Online]. Available: http://www.ens-lyon.fr/ensnano/

[12] S. Li, Q. Jiang, S. Liu, Y. Zhang, Y. Tian, C. Song, J. Wang, Y. Zou, G. J. Anderson, J.-Y. Han *et al.*, “A DNA nanorobot functions as a cancer therapeutic in response to a molecular trigger in vivo,” *Nature biotechnology*, vol. 36, no. 3, pp. 258–264, 2018.

[13] C. Maffeo and A. Aksimentiev, “MrDNA: a multi-resolution model for predicting the structure and dynamics of DNA systems,” *Nucleic Acids Research*, vol. 48, no. 9, pp. 5135–5146, 03 2020. [Online]. Available: https://doi.org/10.1093/nar/gkaa200

[14] M. J. Patitz, “An introduction to tile-based self-assembly and a survey of recent results,” *Natural Computing*, vol. 13, no. 2, pp. 195–224, 2014.

[15] E. Poppleton, J. Bohlin, M. Matthies, S. Sharma, F. Zhang, and P. Šulc, “Design, optimization and analysis of large DNA and RNA nanostructures through interactive visualization, editing and molecular simulation,” *Nucleic Acids Research*, vol. 48, no. 12, pp. e72–e72, 05 2020.

[16] “time — Time access and conversions,” Python Software Foundation. [Online]. Available: https://docs.python.org/3/library/time.html

[17] P. W. Rothemund, “Folding DNA to create nanoscale shapes and patterns,” *Nature*, vol. 440, no. 7082, pp. 297–302, 2006.

[18] “Slurm Commercial Support and Development,” SchedMD. [Online]. Available: https://www.schedmd.com

[19] N. C. Seeman, “Nucleic acid junctions and lattices,” *Journal of Theoretical Biology*, vol. 99, no. 2, pp. 237–247, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022519382900029

[20] ——, “An overview of structural DNA nanotechnology,” *Molecular biotechnology*, vol. 37, no. 3, pp. 246–257, 2007.

[21] N. C. Seeman and H. F. Sleiman, “DNA nanotechnology,” *Nature Reviews Materials*, vol. 3, no. 1, pp. 1–23, 2017.

[22] “CanDo – Computer-aided engineering for DNA origami,” Software Freedom Conservancy. [Online]. Available: https://cando-dna-origami.org

[23] “Git,” Software Freedom Conservancy. [Online]. Available: https://git-scm.com/

[24] “UCSF Chimera an Extensible Molecular Modeling System,” UCSF Resource for Biocomputing, Visualization, and Informatics. [Online]. Available: https://www.cgl.ucsf.edu/chimera/

[25] “AHPCC Resources,” University of Arkansas. [Online]. Available: https://hpc.uark.edu/hpc-resources/index.php

[26] “OxDNA,” University of Oxford. [Online]. Available: https://dna.physics.ox.ac.uk/index.php/Main_Page

# 7 Appendix – README

The following is the content of the README file that provided in the GitHub repository along with oxdna-tile-binding.

## 7.1 Setting Up oxdna-tile-binding

1. Clone the repository
2. Run `cd oxdna-tile-binding`
3. Run `module load python/3.6.0-anaconda`
4. Run `echo ". /share/apps/python/anaconda3-3.6.0/etc/profile.d/conda.sh" >> ~/.bashrc`
5. Run `echo "conda activate" >> ~/.bashrc`
6. Set the paths in oxdna-tile-binding/config.yml for your system
7. Run `conda activate pyoxdna` (make sure your conda environment is set to pyoxdna every time you use these scripts. Sometimes, if you log off of a server, it may reset your conda environment, and you will have to run this command again)

If you run into any issues running things locally (with errors like `KeyError: 'PYOXDNA_HOME'`), you may need to set the environment variables by following these instructions:

- If you want to temporarily set your environment variables (until you exit) run `export ENV_NAME=PATH` (e.x. export `OXDNA_HOME=/home/username/oxDNA`) on each variable

- If you want them to always be set you can run `echo "export OXDNA_HOME=/home/username/oxDNA" >> ~/.bashrc` on each variable OR open `~/.bashrc` with an editor and add these lines at the bottom:

```
export HOME=/home/username
export SIM_HOME=/home/username/simulations
export OXDNA_HOME=/home/username/oxDNA
export PYOXDNA_HOME=/home/username/oxdna-tile-binding/pyoxdna
```

(make sure you change the paths to the correct ones for your system)

## 7.2 Setting up oxDNA

oxDNA[1] is a DNA molecular simulator and analysis tool developed at the University of Oxford. Good documentation can be found here[2].

### 7.2.1 Setup

*Notes before you start*:

- You may need to change the version of gcc
- in select_compute_arch.cmake replace 4 instances of "VERSION_GREATER_EQUAL" with "VERSION_GREATER"

Follow these[3] directions. We used these following commands to compile:

```
cd oxDNA

mkdir build

cd build

module load cuda/9.2 cmake gcc/9.1.1

cmake -DCUDA=1 ..

make
```

### 7.2.2 Using oxDNA

#### oxDNA File Formats

oxDNA has 4 main files:

- An input parameter file (usually named "input" or "input.params" or "input.dat"). This specifies all of the parameters for the simulation, including the type of simulation, the number of simulation steps, the temperature of the simulation, how often to print output, etc.

[1]<https://dna.physics.ox.ac.uk/index.php/Main_Page>
[2]<https://dna.physics.ox.ac.uk/index.php/Documentation>
[3]<https://dna.physics.ox.ac.uk/index.php/Download_and_Installation>

- An input topology file (top file). This specifies the bases and topology of the DNA strands to be simulated. This tells the program which nucleotides are present in the simulation and how they are connected. Documentation here[4].
- An input configuration file (conf file). This specifies how the nucleotides defined in the topology file are oriented in space. Documentation here[5].
- An output trajectory file. This is a giant file showing the position and orientation of all the DNA strands at each recorded timestep during the simulation. It's a plain-text file containing multiple configuration files separated by newlines. You can change how often the configuration is recorded by modifying the print_conf_interval input parameter.

Because each of these file formats is plain text and contains only decimal numbers, it is possible to create and manipulate these files without interacting with external programs, such as scadnano. For example, it's possible to generate custom (or random) configuration and topology files or to splice configurations together. Examples of this can be found in create_tiles.py and computation_experiment.py. Although it's a huge file, pyoxdna/analysis/base.py, the python code that comes with oxDNA, is a good reference for file formats and file modification.

A great visualization tool for oxDNA simulations is Oxdna-viewer[6]. To load a simulation, click "open" and then select either a top and conf file OR a top and trajectory file.

## 7.3 Scripts and Packages in oxdna-tile-binding

### 7.3.1 Pyoxdna

A python package written by Kyle Sadler to interact with oxDNA in python. It has two important modules:

- **pyoxdna** is the main module for running simulations in oxDNA. Given a direc-

[4]`<https://dna.physics.ox.ac.uk/index.php/Documentation#Topology_file>`
[5]`<https://dna.physics.ox.ac.uk/index.php/Documentation#Configuration_file>`
[6]`<https://sulcgroup.github.io/oxdna-viewer/>`

tory and a python dictionary containing oxDNA input options[7], pyoxdna will run an oxDNA simulation. Documentation can be found in pyoxdna/pyoxdna.py and example usage can be found on line 159 in run_simulations.py

- **dna_relaxer** is a module for automatically relaxing DNA configurations. DNA configurations must be relaxed in order to simulate them in oxDNA. Here[8] are some notes on relaxation. Documentation for DNARelaxer can be found in pyoxdna/dna_relaxer.py and example usage can be found on line 18 in run_simulations.py

### 7.3.2 utils/job_launcher.py

This is a python module for easily launching jobs on Razor. Given a command, working_directory, dependencies, job file, etc., this module will automatically create a job script and submit a job to your computing cluster on your behalf. Documentation can be found in job_launcher.py and example usage can be found on line 120 in run_simulations.py.

Many scripts allow the user to input a job_file.

The job input file should have the command the script should be run with in the first line (for example `sbatch` for Slurm or `qsub` for PBS/Torque). In the following lines, users should put the job script they wish to run the program with. This job script should only have the system specific parameters including `[job_name]` and `[out_file]` as flags. All other job specific parameters (such as loading modules or running commands) will be formatted by job_launcher.py, and `[job_name]` and `[out_file]` will be replaced with the job appropriate data.

Example of an input job script:

```
sbatch
#!/bin/bash
#SBATCH --job-name=[job_name]
#SBATCH -p gpu72
#SBATCH --nodes=1
```

[7]<https://dna.physics.ox.ac.uk/index.php/Documentation#Input_file>

[8]<https://docs.google.com/document/d/1zP__47jWaXR0NSNC0wEH4XGCFSHxNVxmBai3VXTGlN0/edit>

```
#SBATCH --ntasks=6
#SBATCH -t 72:00:00
#SBATCH -o [out_file]
cd "/scratch/$SLURM_JOB_ID"
```

The job.txt that results when used as input to job_launcher.py (which will be run with `sbatch`):

```
#!/bin/bash
#SBATCH --job-name=example1
#SBATCH -p gpu72
#SBATCH --nodes=1
#SBATCH --ntasks=6
#SBATCH -t 72:00:00
#SBATCH -o /home/user/example1/example.out
cd "/scratch/$SLURM_JOB_ID"
module purge
module load mkl/18.0.2
module list
export HOME=/home/user
export PATH=$PATH:/home/user/oxDNA/build/bin
time python3 /scrfs/storage/user/home/example.py
```

### 7.3.3 analyze.py

This is a script for analyzing tile binding in oxDNA after simulation. Given input, topology, and trajectory files, analyze.trim_strands() creates a topology and trajectory files with ONLY the strands that bind during the simulation. This makes it easier to see strand interaction in a large simulation.

*Run with*: `python analyze.py [input_file] [trajectory_file] [top_file] [job_file]`. If you would like to run locally, run without a job_file.

Since we can compute simulation binding events (e.g. nucleotide1 binds to nucleotide2 at time t), this script can be improved to filter for bonds with x or more bound nucleotides. Therefore, this can be used to automatically count the number of full tile bindings in a simulation.

Credit: this script is based on the work of Michael Sharp who worked on a similar project with Dr. Patitz a few years ago.

### 7.3.4 computation_experiment.py

A script to profile the run time of oxDNA's simulations using both the GPU and CPU based on the number of nucleotides in the simulations. Results from the experiment are found in [HOME]/results.txt

*Run with*: `python computation_experiment.py run [num_strands (required)] [output_dir (optional, default is SIM_HOME])]` to run the experiment with a specific number of strands or `python computation_experiment.py [start_num_strands] [end_num_strands] [0 for CPU 1 for GPU] [job_file]` to run the experiment with values in the range of `start_num_strands` to `end_num_strands`.

We recommend that the user run a range of sizes multiple times to determine which sizes of structures are best suited for GPU or CPU use.

### 7.3.5 create_tiles.py

A script to generate a grid of aligned tiles for simulation. This works by "copying and pasting" a conf and top file of two complementary tiles in order to make an x by y grid. Input files for this program can be found in oxdna_files/tiles/original. Output files can be found in oxdna_files/tiles.

*Run with*: `python create_tiles.py run [file_name] [num_tiles] [output_dir]` to run

### 7.3.6 run_analysis_multi.py

A small script to run other scripts for multiple iterations. Uses run_analysis.txt as input

*Contents of run_analysis.txt*:

```
First line: number of iterations to run
Second line: the string to run
```

*Run with* `python run_analysis_multi.py`

### 7.3.7 run_simulation.py

This program runs a simulation given a config file, topology file. There are several options the user can choose for relaxation and simulation.

*Run with*: `python run_simulation.py -t [top_file] -c [conf_file] -j [job_file] -r` (this command will just preform relaxation)

Only a topology file, a config file, and either the -r or -m flag are required for run_simulation.py . To run locally, simply exclude the -j flag and the job_file. To see more configuration options, run `python run_simulation.py -h`.

Depending on your system, the output files may end up in a temporary folder if you run this script as a job. Be sure to move those to a more permanent location before they expire.

### 7.3.8 tile_binding_auto_monitoring_light.py

This script runs and analyzes tile binding simulations to see if tiles will attach or detach.

*Run with*: `python tile_binding_auto_monitoring_light.py -s [sim_conf_file] -b [bonds_file] -i [num_iterations] -o [out_dir]`. Only the simulation configuration file, bonds file, output directory, and number of iterations to run are required. To see more configuration options, run `python tile_binding_auto_monitoring_light.py -h`

An example of a simulation configuration can be found as a result of running run_simulation.py or in `pyoxdna/configs/molecular-dynamic.conf` (though this file is missing a topology and conf_file entry).

The `bonds_file` must have the .py file extension, and should look like the example as follows:

```
STARTING_BONDS = [
    [944, 498],
    [945, 497],
    [946, 496],
    [947, 495],
    [948, 494]
]

TARGET_BONDS = [
    [949, 576],
    [950, 575],
    [951, 574],
    [952, 573],
    [953, 572],

    [954, 456],
    [955, 455],
    [956, 454],
    [957, 453],
    [958, 452],

    [959, 639],
    [960, 638],
    [961, 637],
    [962, 636],
    [963, 635],
]
```

Where each `[ID1,ID2]` is a pair of IDs of a nucleotide in the topology file. The program exits when **all** of the `STARTING_BONDS` have dissolved (printing `THE TILE DETACHED!`)

or **any** of the `TARGET_BONDS` have formed (printing `TILE IS ATTACHED!`).