MONTE CARLO SIMULATION OF DNA ORIGAMI SELF-ASSEMBLY

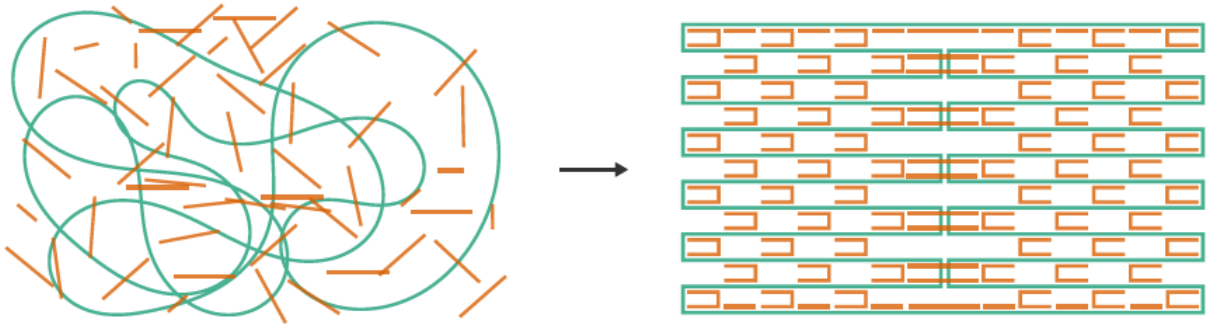
Notes: Jakub Lála (jl24018@ic.ac.uk)

Model: Alexander Cumberworth (alexandercumberworth@gmail.com)

DNA ORIGAMI

DNA nanotechnology has its origins all the way in the 1980s, when it was first proposed that the double helix structure may be used as a construction material for the assembly of geometrically defined objects with nanoscale features. However, the invention of the DNA origami scaffold-based method comes only in 2006 with Rothemund's ground-breaking article in the Nature.

In DNA origami, a single-stranded DNA strand referred to as *scaffold* is folded using short oligonucleotides called *staples*. It is a bottom-up assembly, meaning the construction components inherently contain the information of the folded target structure and thus the reactants self-assembly given only sufficient time and conditions. Below in the figure, a schematic of the assembly is shown.



One of the applications of DNA origami may be a viral detection mechanism. If one knows the exact gene sequence of a virus, one may design staples in such a way that given specific conditions, the virus' gene information folds. The folded origami may then be detected using microscopy, therefore, identifying the presence of the virus in a patient's blood.

ALEX'S MODEL

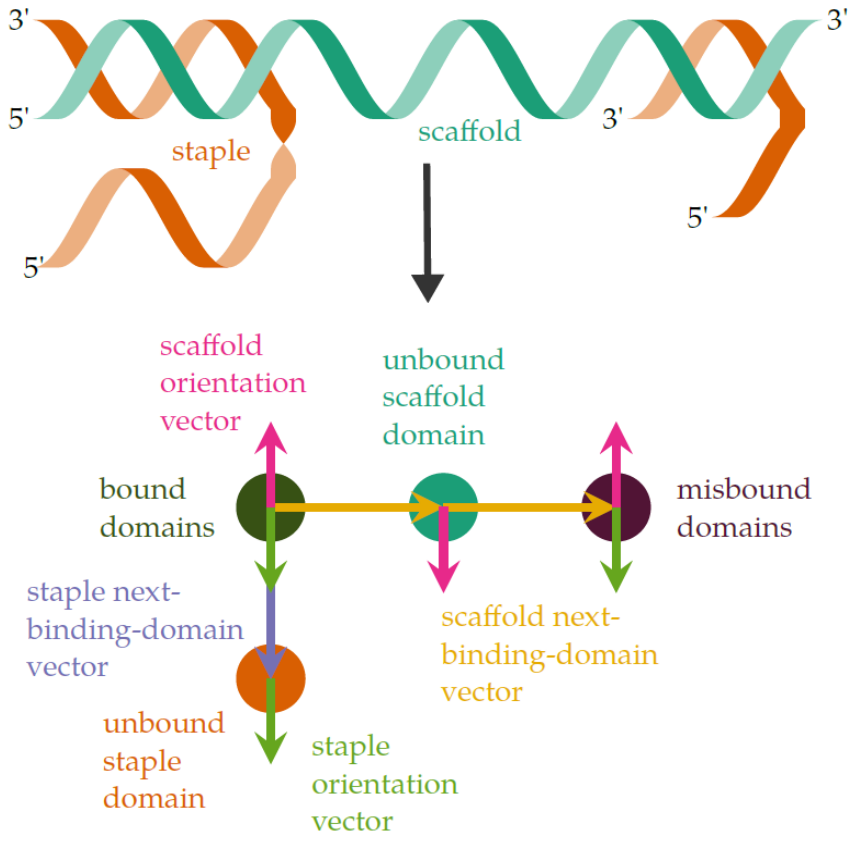
The model comes from Alex's article and thesis project. Its aim is to find a compromise between the physical accuracy of oxDNA models, where individual nucleotide molecules are being simulated leading to demanding computation, and the efficiency of statistical models, where, however, the lack of explicit geometry of the system results in strong assumptions.

Particles on a lattice

The model is designed to represent parts of the single-stranded DNA strands as particles on a *simple cubic lattice* with 90 degrees between helical axes. This approximation has a good precedent in previous studies of DNA bricks and nucleation kinetics.

Particles are defined to be binding domain units of several nucleotides. Currently, the model supports a HalfTurn and ThreeQuarterTurn domain classes, where the class description is based on the fraction of a turn remaining after an integer amount of whole turns that make up the double helix. For example, a 16-nucleotide binding is composed of 1.5 turns of the helix structure, hence that would be defined as the half-turn domain type. Note that larger binding domains must be integer multiples of the smallest domain unit, as well as that domains with one or two residue nucleotides may be approximated to an integer binding domain as the relative length difference is small.

On a single lattice point, the occupancy may either be 0 (unoccupied), 1 (unbound) or 2 (bound or misbound), where the number indicates how many binding domains (particles) are present. A bound state is defined only when the domains are fully complementary in terms of their gene sequence. The figure below shows a schematic of transformational description between the double helix and particle representation.



To account for the restrictions imposed on the DNA chain by the double helix, two types of vectors are defined for each particle – *orientation vector* and *next-binding domain vector*, both shown on the figure above. The need for such describing tools is to set requirements that allow crossovers between strands to occur only at certain intervals, as well as transmit information on the current phase of the helix. Directions are defined to be positive for a scaffold chain going from a helical position of 5′ to 3′, whilst for a staple chain it is from 3′ to 5′, as shown on the figure.

The orientation vector points orthogonally from the helical axis to the position of the strand at the end of the helix in the current binding domain. In the scaffold at the top of the figure, when going in the positive direction as defined above, one sees that at the end of the first binding domain, the scaffold is at the top, and hence the scaffold orientation vector for that binding domain must point upward. Note that in an unbound state, the direction of the vector is uniformly distributed. Also note that in a (mis)bound state the sum of these vectors must add up to zero, i.e. they must point in the opposite directions.

The next-binding domain vector connects two binding domains pointing to the next domain along the given chain. This vector is crucial for implicitly resolving the helical axis of a domain, as no explicit helical axis vector is defined. For a single pair of binding domains (on a single lattice site), the helical axis remains to be constrained only in a plane rather than an actual single direction. Moreover, only contiguous binding domains are allowed on adjacent lattice sites.

Grand-canonical ensemble

A single *scaffold* is being simulated in the *grand-canonical ensemble*, i.e. the chemical potential of the staples is held constant, whilst the fluctuating number of staples is observed. That means that only the staples (mis)bound to the system are of interest.

Although staple-staple binding is not common in the relevant simulated temperatures, as the staples are not designed to bind with each other, the model allows for staple-staple binding due to a potential local staple concentration increase at a scaffold. Nevertheless, the staples must still be (mis)bound to the scaffold – either directly or indirectly via another staple bound to the system.

The key parameter is the availability of the staples to the system, which is determined by the initial staple concentration, binding state of staples on the scaffold and binding of staples to other staples. Also note that in actual experimental DNA assembly protocols, the staples are in excess, hence the free staple concentration is assumed to be constant regardless of the staple binding states.

The model takes only a few input parameters – temperature, salt concentration (in which the self-assembly takes place), staple

concentration and stacking energy (see stacking term in the Hamiltonian section below)

Hamiltonian

The Hamiltonian, energy function of the system, considers the potential energy of the system, which has bonding, stacking and steric contributions.

The bonding term comes from the hybridization free energies associated with (mis)bound states using the SantaLucia NN (nearest neighbour) model. This model also in a coarse-grained manner accounts for the entropy of hybridizing two molecules. Moreover, this model has to be corrected using the mean field correction as in to adjust for the model's inadequacy in terms of entropic deviations. This comes from the investigation of the same system but with differently sized binding domains, which yielded contradicting results. Note, however, that one may also adopt a uniform or sequence-specific definition of the hybridization energy.

The stacking energy is to some extent simplified but allows for both stacked and kinked configurations. Formally, kinked configurations were disallowed which showed very difficult sampling, where the system had to be rearranged through domains unbinding and rebinding, increasing the amount of simulation steps.

Monte Carlo

Monte Carlo (MC) simulations are fundamentally attempting to randomly sample a given phase space with a probability distribution function to estimate a quantity of interest. Those are thermodynamic or kinetic properties defined as averages over the configuration space of a given system as a function of the control variables of the selected statistical ensemble. In this model, it is the chemical potential of the staples (more specifically their concentration) that we set as the control variable, effectively fixing it.

The phase space is usually sampled by importance sampling (i.e. the Metropolis algorithm), where a random trial configuration is generated and is accepted according to some acceptance probability. Such acceptance probability must be devised in such a way that the Markov chain of independent conformations obeys detailed balance.

In order to efficiently sample the configuration space of the given system, advanced biasing schemes must be employed in order to steer sampling in the relevant regions of the phase space with a significant contribution to the result.

The simulation may be run in several modes – constant temperature (traditional Monte Carlo simulation), parallel tempering (or REMC) and umbrella sampling.

Parallel Tempering

As low temperature simulation setups may become trapped in a local minima in the phase space, their sampling of the phase space may be quite challenging and demanding on a relevant timescale. On the other hand, high temperature simulations are able to sample large volumes of the phase space. Therefore, a method has been developed to allow for low temperature simulations to access a representative region of the phase space by permitting information exchange between simulations at various temperatures.

The method is replica exchange MC simulation (REMC or parallel tempering) and it is based on the idea of simulating multiple replicas (copies) of a simulation simultaneously. These different replicas differ only with respect to some simulation control parameter, most commonly the aforementioned temperature. These variables are referred to as exchange variables.

Between a random pair of these replicas a random exchange swap attempt occurs at a random step interval, provided that the replicas are adjacent with respect to the exchange variable. An accepted exchange leads to a complete exchange of their configurations. This model specifically performs an exchange attempt at pre-defined step interval, alternating between an exchange of even and odd pairs of replicas. Although this does not obey detailed balance, the algorithm still obeys total balance and so is valid.

Note that in this model, when temperature is chosen as the exchange variable, the chemical potential of the staples must also be exchanged, yet as it is temperature dependent, the REMC is still defined by a single exchange variable. Note, however, that two-dimensional variations of REMC may be used. The model already implements this possibility, where the exchange variables may be whichever of the following – temperature, stacking energy or system biases.

By running on a community cluster, one may take advantage of the CPU and run the replicas parallel. To optimise, one must ensure that the highest temperature simulated is high enough to ensure the volume sample is large enough to access all relevant regions of the phase space, as well as the number of replicas should be chosen such that swapping occurs between all adjacent replicas (see paper for more in depth optimisation suggestions).

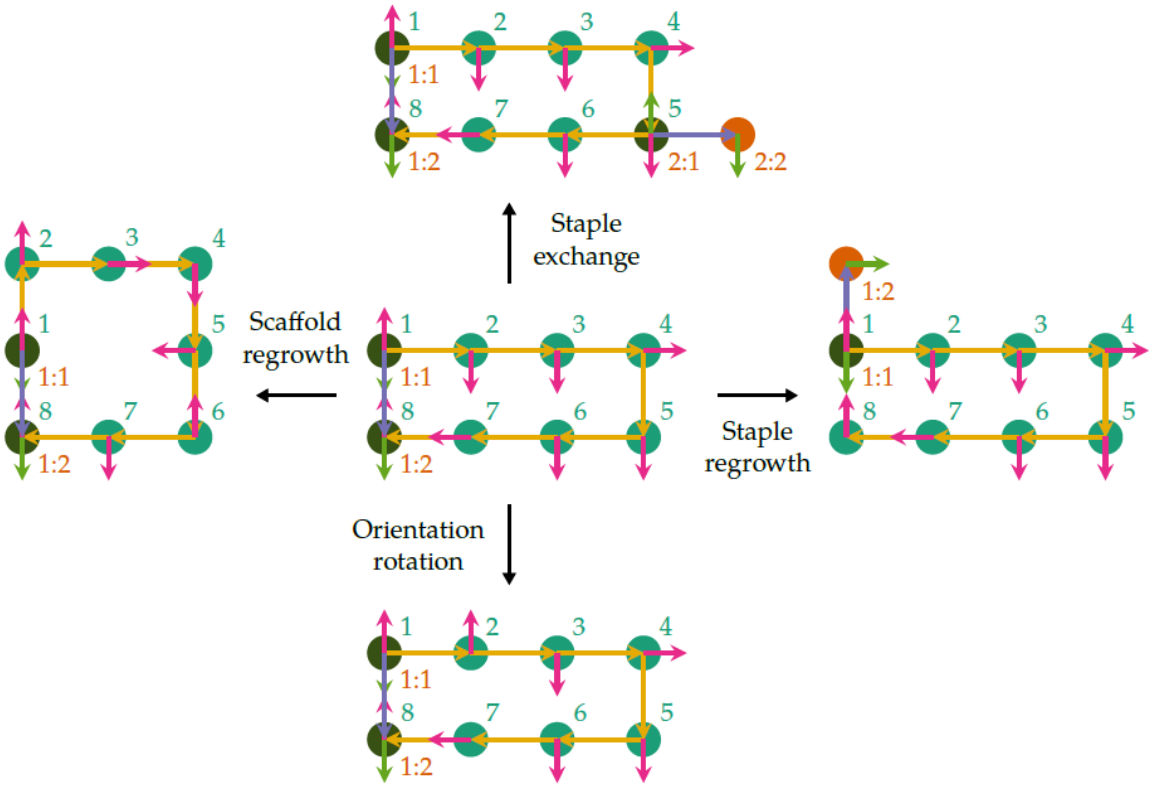
Move Types

Due to specific and strong interactions of this model, efficient sampling of near-assembled states is quite difficult using traditional MC methods for polymeric systems.

Special schemes for re-growing domain units is introduced, resulting in three possible regrowth variants that may be used:

- *symmetric* – uniform probability of choosing any position difference and orientation vector, meaning the classic canonical acceptance criterion is applied
- *configurational bias* (CB) – essentially Rosenbluth sampling, where bias is introduced with respect to the associated energy change between new and old configuration trials
- *recoil growth* (RG) – growth of each binding domain selects a configuration with a uniform probability and is either labelled *open* (can and is used for the successive binding domain) or *closed* (cannot be used and another is proposed up to a total of k_{max} trials); if no open configurations occur, growth recoils to the most recent binding domain, testing new open configurations; recoiling can occur l_{max} times or until all binding domains being grown are unassigned (closed) and thus the move is rejected

Any of these re-growth variants may be used in any of the steps, although it is always mentioned what type of biasing re-growth variant is employed in the code.



The figure above shows the four different move types used in the model. This figure will be referred to as the example figure later on.

Orientation Vector

The most simple move type, where a random orientation vector has its direction changed according to the following:

- select a random binding domain
- generate a new orientation vector with uniform probability

For unbound domains, the change in energy due to the orientation vector change is zero, hence acceptance probability is always unity and the move is accepted. For (mis)bound domains, the partner domain's orientation vector must be also modified to ensure that their sum remains zero. Therefore, the additive inverse of the former domain's orientation vector is used for the partner domain.

In the example figure, the orientation vector for scaffold domain 2 is changed from downward direction to upward direction.

Staple Regrowth

The next two move types only concern themselves with the staples. The regrowth move regrows a random staple according to:

- select a random staple with uniform probability
- select a random (mis)bound domain of the staple
- regrown the staple from the domain in both directions using CB

If no staples are present the move is rejected immediately. Also, if a connecting staple is selected that would after removal lead to a network of staples not connected to the system (to the scaffold), then the move is also automatically rejected.

In the example figure, the staple is re-grown from binding domain 1:1, hence that one remains constant between the configurations whilst staple domain 1:2 changes its position.

Staple Exchange

This move types considers removing and adding different staple types by:

- decide whether to insert or delete a staple
- select a random staple type
- insert/delete accordingly with symmetric staple growth

As the traditional symmetric method of regrowth is used, the acceptance probability follows the usual grand-canonical acceptance, where the pre-factor deals with the number of ways a staple may be inserted compared to the ways it may be removed.

In the example figure, a staple is inserted on scaffold domain 5 with staple domain 2:1 being the one chosen to (mis)bound to the scaffold.

Scaffold Regrowth

The last move type concerning the scaffold is the most complex. The idea is to sample scaffold conformations independently from binding states of staples. This is because a traditional scaffold regrowth approach of near-assembled states would rarely propose a trial configuration of as many bound domains. As the trial states would commonly be less assembled and would thus have a higher energy (less favourable), the move would almost always result in a low or zero probability, therefore, being rejected.

The staple binding state in this move is thus held constant, i.e. the binding state of staples is altered only in the *Staple Regrowth* and *Staple Exchange* move types. Separating the scaffold move type from the rest improves the acceptance rate but also allows for an easier implementation of new move types. One may view the system as a network, where (mis)bound domain pairs act as nodes. By keeping the staple states constant, the network topology remains constant as well, and thus this move type can be referred to as a *conserved topology* move.

Additionally, the regrowth of the scaffold will be done in a similar fashion to a fixed-end CB regrowth. Such scheme allows polymers to grow to a predetermined endpoint by biasing the selection of each polymer unit configuration by the number of ideal walks from the trial polymer unit's position to the endpoint. In layman's terms, certain endpoint constraints will be defined by the scaffold length, the original configuration, the position of staples on the scaffold, etc. The regrowth of the scaffold is then limited to always comply to these constraints. An example would be when the remainder of the scaffold to be regrown must circle back to a staple to which the scaffold end was bound in the former configuration. At that point, the scaffold is quite constrained to let us say grow in the shortest possible distance to the staple, if the segment to be still regrown has become sufficiently short.

This idea may be extended to multiple endpoints per a segment of the scaffold being regrown. Moreover, the growth of multiple segments of a scaffold on possibly multiple chains may be established. Each segment can then have its own set of endpoint constraints. For a binding domain that acts as an endpoint, but must also be grown, the endpoint constraint is defined to be inactive until the associated configuration of that binding domain has been determined.

At that point, the number of ideal walks is no longer valid for biasing as they could differ for old and new configurations, hence an indicator function is set up – unity if ideal walks remain and zero otherwise. (Note that in the input files of the model an archive file for the number of ideal walks is still used, although the actual code only considers if any ideal walks remain.) Moreover, another indicator function is employed to prevent new pairings of binding domains -

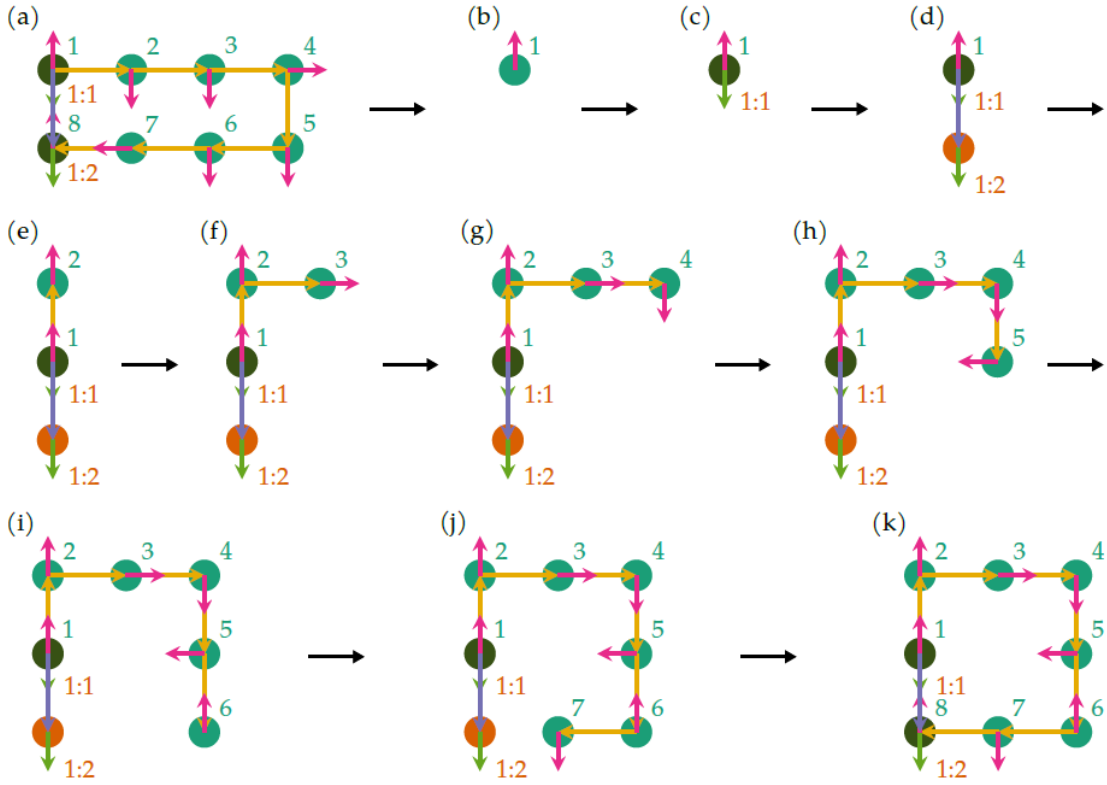
unity for unoccupied lattice, occupied by another binding domain of the chain being grown or the position of an active endpoint on the segment being grown, and zero otherwise.

Also note that misbinding between domains on the same chain is allowed, as Staple Exchange and Staple Regrowth move types do not sample such states involving scaffold domains misbinding with themselves.

This move type can be done for a single segment or multiple segments and generally goes as follows:

- select a single segment or multiple segments
    - select a random seed binding domain
    - select a random segment length and direction
    - keep adding binding domains to the seed binding domain until
        - the selected segment length is reached OR
        - the end of the chain is reached (if at that point the selected segment length has still not been reached, start adding binding domains from the opposite direction of the seed binding domain)
- staples in the segments are determined to either be regrown or acting as endpoints based the staple position
    - *internal staples* – involved in the segment to be regrown and thus are regrown together with the scaffold (note that these staple domains are always regrown before re-growing the scaffold domains whenever a (mis)bound staple domain on the scaffold is reached)
    - *external staples* – not involved in the segment to be regrown and thus remain in their old configuration state with those bound to the scaffold acting as endpoints

An example of a single segment scaffold regrowth move is shown in the figure below. The segment to regrow is chosen to be the entire scaffold, where the seed binding domain is domain 1. Initially, the seed domain is grown. Afterwards the staple domains must be re-grown first before domain 2. Then the domain chain is being successively grown. When scaffold domain 5 is grown, there are only two ways the scaffold could grow to satisfy the endpoint constraint – binding scaffold domain 8 with staple domain 1:2.

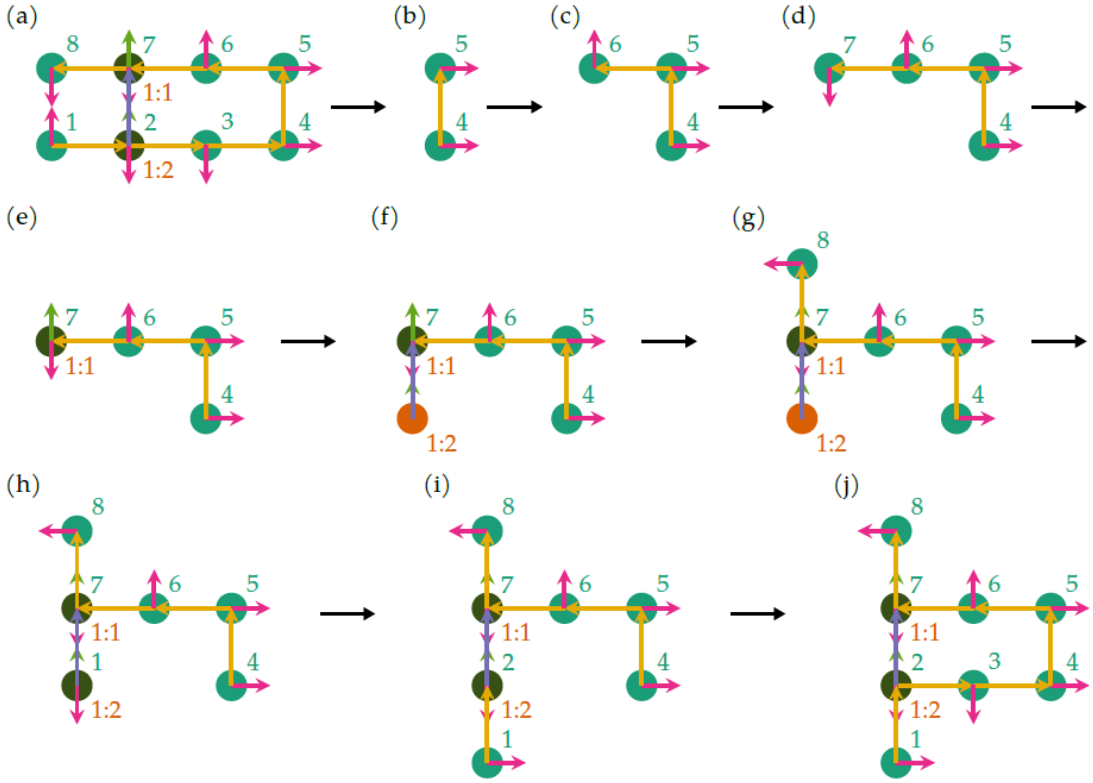


The multiple segment variation of this move type is designed so that the selection of binding domains enables jumping between scaffold domains at points where two scaffold domains are adjacent through a linking staple. Other than that, the approach is very similar to the single segment variation.

Whenever a binding domain is added to a segment for regrowth and is bound to a staple, then if there is another scaffold domain that is neither already selected in the segment, nor contiguous with a scaffold binding domain already in the regrow segment, it is added to a queue of potential segment seed binding domains. Once the selection of domains is terminated for a specific segment (by reaching that particular segment's length) and the maximum total number of binding domains has not been reached, a new segment is created using the binding domain seed from the queue with the direction chosen as with the initial segment. If that one terminates, another segment from the same binding domain seed is started but in the opposite direction. This continues until complete termination when the maximum number of binding domains is reached. Segments are then regrown in the order with which they were selected.

In the figure below, an example of the multi-segment regrowth is shown. Initially, the maximum number of binding domain to be selected is chosen to be seven. Then the first seed binding domain is randomly chosen to be scaffold domain 5, for which the selection direction is chosen to be forward and the maximum segment length is chosen to be three. While scaffold domains 6, 7 and 8 are added to the first segment, scaffold domain 2 is added to the queue for potential next seed binding domains. Once the first segment is completely selected,

the seed is used to begin a new segment at scaffold domain 2. The negative direction is chosen and a segment length of two is selected, resulting in scaffold domain 1 and then 3 to be added (as the other direction for the seed is used after the chain end is reached). Although the maximum number of binding domains was not reached, there are no more potential seeds in the queue, hence the re-growth then proceeds sequentially.



USING THE CODE

The DNA model code was written by Alexander Cumberworth and is originally available in his GitHub repository. There is also a GitHub repository made by Jakub Lála that contains some additional scripts for the specifics of his investigation – the relative importance of staples on the stability of the assembled structure.

Installation

After cloning the repository, two applications have to be made using the relevant Makefiles – **latticeDNAOrigami** and **calc_num_walks** (located in **scripts/ideal_random_walks**). The former is the main application, whilst the latter creates an archive file of ideal random walks used for the scaffold regrowth move type.

There are the library dependencies necessary for compiling and running the program: **boost**, **JSONcpp**, **openmpi**, **make**, **gcc**, **zlib**. Note that the Makefiles may include the *mpicxx* command, which is not available in newer versions of openmpi. Also check that all relevant objects and library are being linked the Makefiles, as well as that the **build/** and **bin/** are created prior to running the Makefile, if an error is being thrown.

Running the simulation

The simulation is run by using the command

- *latticeDNAOrigami -i* [configuration file]

or

- *latticeDNAOrigami -i* [configuration file] > [output file]

depending whether one wants to output into the terminal or into an output file respectively.

Moreover, one may list all the input parameters by the command

- *latticeDNAOrigami -h*

Running on the cluster

Running the code in the command line of the terminal requires for the terminal to remain open, which is not quite useful when trying to run simulations for a few hours. One must thus use the **SLURM** batch job workload manager.

There are prepared template shell scripts that can be run with either **SLURM** or **PBS**. Note that the templates have to be filled in with the respective directories, walltime, etc. This can be edited manually or the shell scripts for creating inputs may be used (e.g. *create_serial_inputs.sh*).

An example of the command for running the shell script for SLURM follows as:

- *sbatch serial_template_slurm.sh*

One may check the status of the job by using:

- *squeue*

Also note that the error and output files are defined to be ***.e** and ***.o** files respectively.

If one decides to cancel his job, he may use the following command with the respective job ID:

- *scancel [Job ID]*

Input Files

The list of the key input files goes as follows:

- **.inp configuration file**
  - includes all the input simulation parameters
  - used as the input file when executing the simulation
  - all the input parameters may be printed out by *latticeDNAOrigami -h* command line help option
- **.json system file**
  - includes all the information on the binding domains of the scaffold and the staples
- **.json move type file**
  - defines the move types used and their frequencies
- **.json bias functions file**
  - defines the bias functions for whichever order parameter
- **.json order parameter file**
  - defines the order parameters to be extracted from the simulation (note: the output of order parameters is specified as an parser argument in the .inp file)
- **.arch number of ideal random walks file**
  - stores the number of ideal random walks of the scaffold for the scaffold regrowth move

Simulation Types

The code enables for 4 different types of simulations:

- *constant_temp* – Constant Temperature
  - most simple type, where a single temperature is chosen, at which the simulation is run
  - only a single parameter:
    - number of MC steps [ct_steps]
- *annealing* – Annealing
  - temperature is linearly being increased during the simulation
  - the simulation starts at a high temperature and is then being reduced according to the parameters
  - it seems as there are no other output files except for the usual **.out**

    - parameters:
        - maximum temperature [max_temp]
        - minimum temperature [min_temp]
        - temperature interval [temp_interval]
        - number of MC steps per temperature [steps_per_temp]
- *parallel_tempering* – Parallel Tempering or REMC
    - replicas of different exchange variables are being simultaneously simulated
    - a prefix is used for the type parameter to define the exchange variable for the swaps between replicas
        - *parallel_tempering* – a general 1D REMC
        - *t_parallel_tempering* – temperature 1D REMC
        - *ut_parallel_tempering* – chemical potential multiplier and temperature 2D REMC
        - *hut_parallel_tempering*
        - *st_parallel_tempering* – stacking energy multiplier and temperature 2D REMC
        - *2d_parallel_tempering* – a general 2D REMC
    - note that the exchange variable for chemical potential, stacking energy and system biases is a multiplier rather than the absolute value
    - parameters:
        - simulation temperatures [temps]
        - number of replicas [num_reps]
        - [swaps]
        - [max_pt_dur]
        - [exhance_interval]
        - [chem_pot_mults]
        - [bias_mults]
        - [stacking_mults]
        - [restart_swap_file]
- *umbrella_sampling* – Umbrella Sampling

Order Parameters

The order parameters to be extracted are defined in the .json file and if specified as an input parameter, they will be print in the **.ops** output file.

The definition of the parameters in the .json file is based on these parameters:

- *label* – complete name of the order parameter
- *tag* - used for defining which order parameters to output in the input file when starting the simulation
- *type* - defines the order parameter type inside the code
    - *Dist* – distance between two domains
        - requires *chain1*, *domain1*, *chain2* and *domain2* specifications

    - *AdjacentSite* – boolean whether domains are next to each other (0 – not adjacent; 1 – adjacent)
        - requires *chain1*, *domain1*, *chain2* and *domain2* specifications
    - *Sum* – sum of other order parameters
        - requires a vector string of order parameters
    - *NumStaples* – number of staples
    - *NumStaplesType* – number of a specific staple type
        - requires staple type specification by its identity from the JSON file
    - *StapleTypeFullyBound* – boolean whether a specific staple type is fully bound (only once by all binding domains)
        - requires staple type specification by its identity from the JSON file
    - *NumBoundDomainPairs* – number of bound domain pairs
    - *NumMisboundDomainPairs* – number of misbound domain pairs
    - *NumStackedPairs* – number of stacked pairs
    - *NumLinearHelices* – number of linear helices
    - *NumStackedJuncts* – number of stacked junctions
- *level*

Output files

All of the type of files below are produced as a result of the simulation, where the filebase is defined by the output filebase given in the **.inp** input file. Note that the ***_vmd** versions of these files describe the final configuration.

- **.counts** – key order parameters evolution
    - [MC step] [Bound Staples] [Unique Bound Staples] [All Bound or Misbound Domain Pairs] [Fully Bound Domain Pairs] [Misbound Domain Pairs]
- **.ene** – system's energy and bias evolution
    - [MC step] [Total Energy] [Enthalpy] [Entropy] [Stacking Energy] [Bias]
- **.moves** – statistics and details of move type acceptances
- **.ops** – order parameters evolution
    - header - [ops1], [ops2], […]
    - body - [MC step] [ops1] [ops2]
- **.ores** – orientation vectors evolution
    - [orientationVector1] [orientationVector2] […]
    - each line describes a single logged MC step
- **.out** – general output file showing simulation progress
- **.staples** – staple count in the system
    - [MC step] [# of staple1] [# of staple2] […]
- **.staplestates** – staple states with respect to their state in the final assembled structure (0 – misbound, 1 – fully bound)
    - [MC step] [staple1] [staple2] […]
- **.states**
- **.trj** – trajectory file containing configuration at the specified MC step

    - [MC step]
      [scaffold identity] [unique ID]
      [domains (x, y, z)]
      [orientation vectors (x, y, z)]
      [staple identity] [unique ID]
      [domains (x, y, z)]
      [orientation vectors (x, y, z)]
      […]
    - can be used to restart the simulation from whichever point in the trajectory
    - this becomes helpful if there is a crash during a simulation or not enough sampling was achieved
    - one may use the trajectory to restart the simulation from a particular configuration and not waste time re-simulating what has already been simulated
- **.vcf** – position of each domain used for VMD visualisation
    - ["timestep"]
      [x1] [y1] [z1]
      [x2] [y2] [z2]
      [x3] [y3] [z3]
      […] […] […]

      ["timestep"]
      [x1] [y1] [z1]
      [x2] [y2] [z2]
      [x3] [y3] [z3]
      […] […] […]

      […]
- **.vsf** – identification and type of domains for VMD visualisation

Visualisation

It was recommended to use **VMD** rather than the LaTeX library **Tikz** by the author of the code. Consider the **.tcl** files in **/scripts/vmd/**. One has to first, however, download the VMD from the official website. For the download, it is necessary to register on the website.

- **liborigami.tcl** – specific library for visualisation of DNA origami from this model
- **pipe.tcl** – live visualisations
    - input arguments – [VMD file directory] [Filebase Name] [Staple Length]
- **view_origami_coors.tcl** – visualisation after simulation completion
    - set the variables inside the VMD Tk Console
        - *libdir* – directory of the library
        - *filebase* –
        - *system* – name of the simulated system
        - *staplelength* – length of staples in the units of binding domains (e.g. staple length = 32 / 16 = 2)

To visualise live as the simulation is running, one has to run the **pipe.tcl** script by using the bash command:

- *vmd -e $(libdir)/pipe.tcl -args $(libdir) $(filebase)_vmd $(staplelength)*

where *$(libdir)* is the directory that the vmd scripts are located, and *$(filebase)* includes the directory that you output the files to.

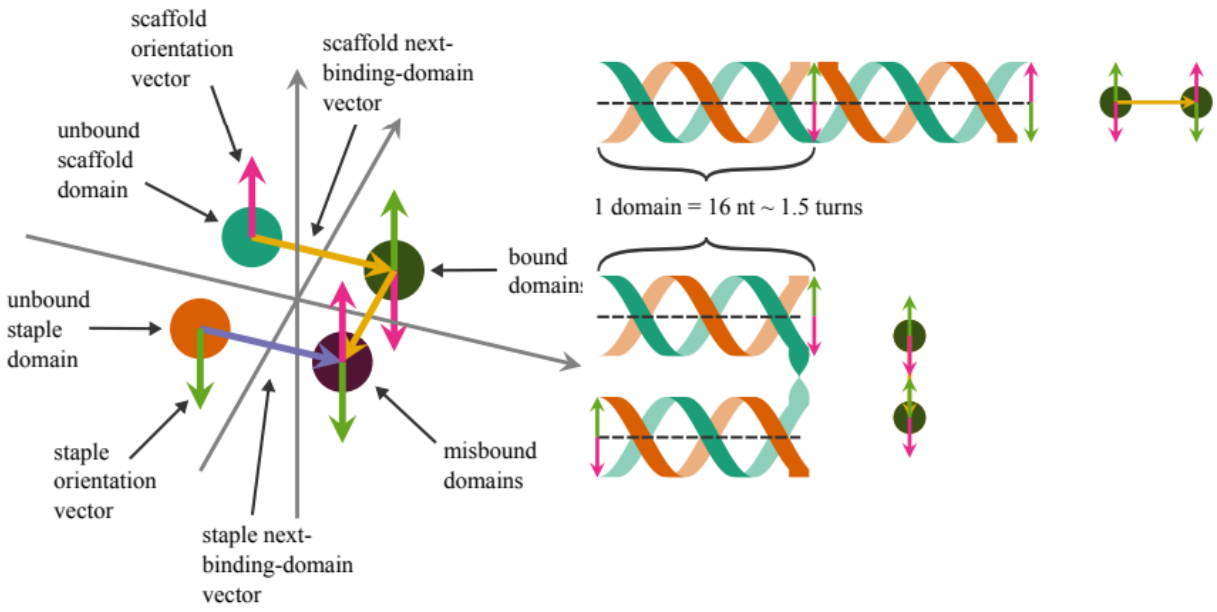
The **filebase_vmd** files are used for live viewing, where the create vmd instance and vmd pipe freq input parameters launch VMD for live viewing and define the frequency of rewriting the vcf and vsf files (i.e. the frequency of updating the live visualisation) respectively.

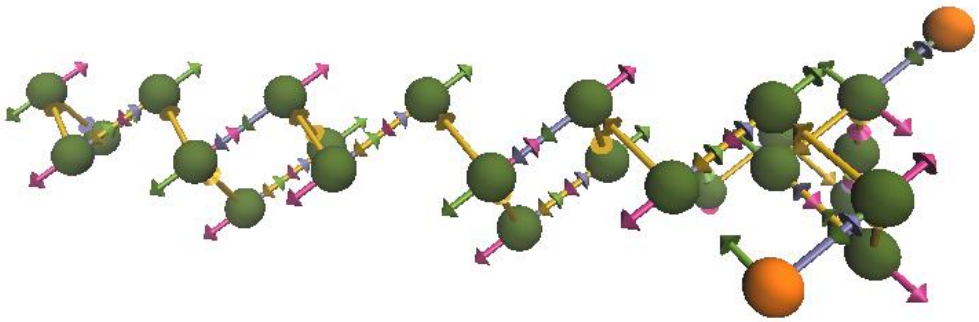
To visualise after the simulation has finished, one can run the **view_origami_coors.tcl** script in the VMD Tk Console accessible from [Extensions >> Tk Console]. After setting the four variables necessary, the .tcl script has to be sourced. Note that for a serial simulation, the *filebase* and *system* variables are the same. When sourced, one may use the VMD control panel to play through the evolution of the simulation, adjusting speed, camera angle, projection settings, etc.

The visualisations' theme is identical to the one in Alex's paper and thesis. The colour description goes as follows:

- **particles**
  - **veal** – unbound scaffold domain
  - **orange** – unbound staple domain
  - **dark green** – bound domains
  - **purple** – misbound domains
- **vectors**
  - **yellow** – scaffold next-binding domain
  - **blue** – staple next-binding domain
  - **light green** – staple orientation
  - **pink** – scaffold orientation

The figure below shows the model description from the paper for completeness and an example of the VMD visualisation respectively.





Analysis

There are several analysing Python scripts included in the **/scripts/analysis** folder. They al take in different arguments, hence one should always use the *--help* to see the necessary inputs.

Note that some of the arguments are not necessary for some simulation setups. Also, apart from the usual modules (numpy, scipy, matplotlib, pandas), **pymbar** and **origamipy** are required, the latter being a specific package made by Alex. Note that pymbar only supports up to Python 3.6, thus it is recommended to use a Python 3.6 environment. The origamipy package may be installed by using the setup.py and running

- *python setup.py develop*

The analysis script are described as follows:

- **calc_domain_occupancies.py** – calculates scaffold domain occupancies for a given simulation set
  - I am not sure what JSON file should be the input…
- **calc_end-to-end_distance.py** – calculates end-to-end distance evolution into a file

    - *system_filename* [snodin_unbound.json]
    - *traj_filename* [test_const.trj]
    - *out_file* [end-to-end-distances_test_const.txt]
- **calc_marginalized_expectations.py** – calculates the mean
    - *system_filename* [snodin_unbound.json]
    - *filebase*
    - *input_dir*
    - *output_dir*
    - *temp*
    - *staple_m*
    - *stack_ene*
    - *tag*
    - *assembled_op*
    - *staple_types*
    - *scaffold_domains*
    - *--reps*
    - *--temps*
- **calc_metlting_temp_lfes.py**
- **calc_numfullyboundstaples.py**
- **calc_radius-of-gyration.py**
- **calc_scaffold_distances.py**
- **calc_scaffold_rmsd.py**
- **extract_configs.py**
- **perform_stack_decorrelation.py**

Plotting

Plotting Python scripts are prepared in **/scripts/plotting** using the **matplotlib** package. Descriptions of each follows:

- **plot_all-lfes.py**
- **plot_barrier-vs-sysvar.py**
- **plot_barrier-vs-temp.py**
- **plot_combo-lfes.py**
- **plot_domain-frequencies.py**
- **plot_domain-melting-temps.py**
- **plot_frequencies.py**
- **plot_means.py**
- **plot_melting-lfes.py**
- **plot_ops_series.py**
- **plot_single-staple-curves.py**

INVESTIGATION

An undeveloped investigation of the relative importance of individual staples was started by Jakub Lála.

Initially, a single-removal (SR) type of simulation was run, where a single staple type is removed from the input parameters and thus is not inserted into the system. The pipeline for creating the input files was developed using Python scripts and is located on Jakub's GitHub forked repository in **scripts/investigation/single-removal/**:

- **create_sr-seq.py** – considers the input sequence file for the staples and iteratively creates sequence files, where a single type is removed
- **create_sr-jsons.py** – considers the input sequence staple and scaffold files and creates the JSON input file of the system for all simulation setups
- **create_sr-input-configs.py** – creates the .inp input files for all simulation setups according to the provided template
- **create_sr-slurms.py –** creates the .sh SLURM files necessary for cluster execution for all simulation setups
- **create_sr-folders.py** – prepares all the necessary files for all simulation setups into a sim_folders/ folder
- **create_sr-all.py** – runs all of the scripts above successively

Note that for proper execution one has to include the necessary input files in the **inps/** folder:

- **(bias_functions.json)**
- **input_template.inp**
- **movetypes_default.json**
- **num_walks.arch**
- **ops_default.json**
- **serial_template_slurm**
- **snoddin_scaffold.seq**
- **snoddin_staples.seq**

The files above are examples, thus if one wants to use the scripts with other DNA origami shapes other than the Snoddin tile, or wants to use different move type frequencies, one has to adjust the filenames accordingly in the script.

Taking the Snoddin example shown below, 13 different simulation setups were performed, where a different staple type was missing in each and one simulation setup had all of the staple types present. The figure below also colourfully groups staple types of similar characteristic, more specifically melting point. This comes from Alex's paper, where he studies the mean average occupancy for individual staple types in this Snoddin tile by performing REMC simulation. By analysing this order parameter for various temperatures, he was able to find the transition and identify it as the melting temperature. Various melting temperatures of staple types

then refer to various degrees of geometrical restrictions on the system.

More specifically, the staple type categories are:

- **same helix** - staples 1 and 12 => most stable, highest T melt
- **span-2** - staples 3, 6, 9
- **span-0, inside span-2** - staples 4, 7, 10
- **span-0, outside span-2** - staples 2, 5, 8, 11 => lowest T melt, thus most geometrically and physical scaffold restrictions

These are listed in the order of decreasing temperature, and thus increasing scaffold geometry restriction.

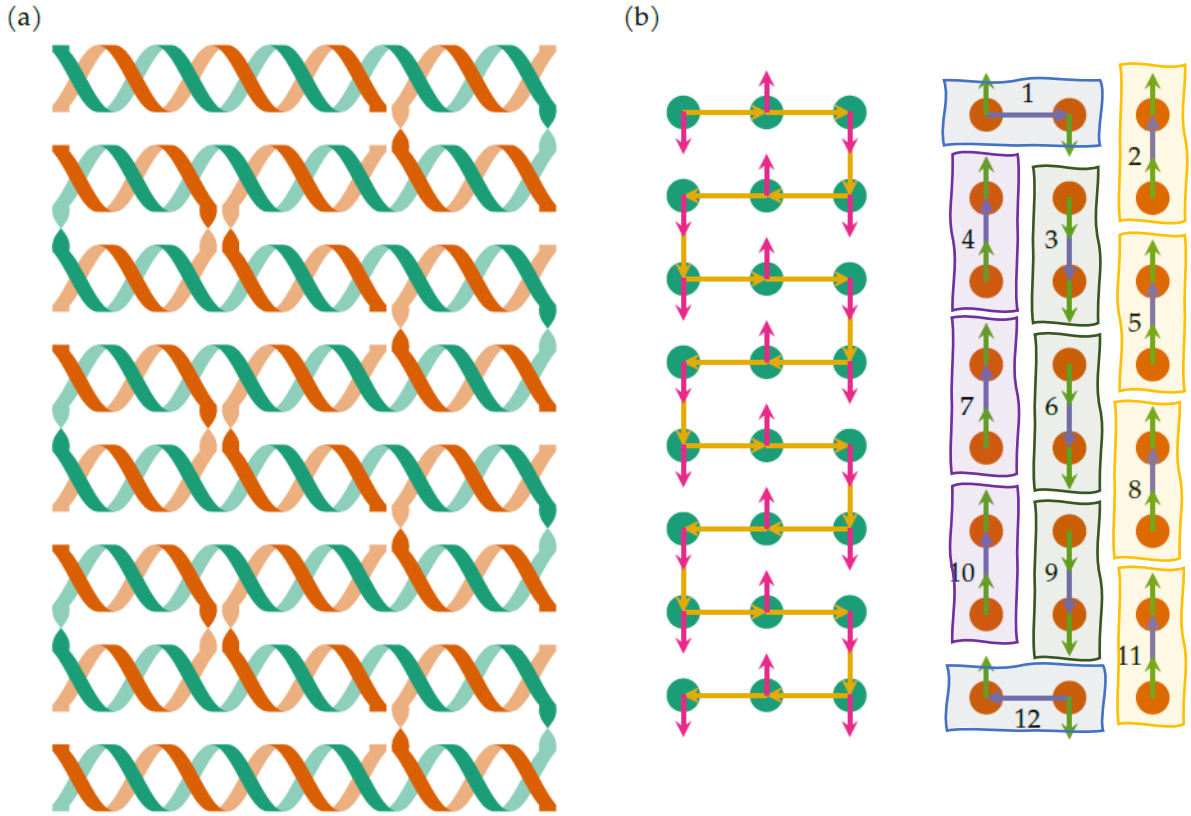
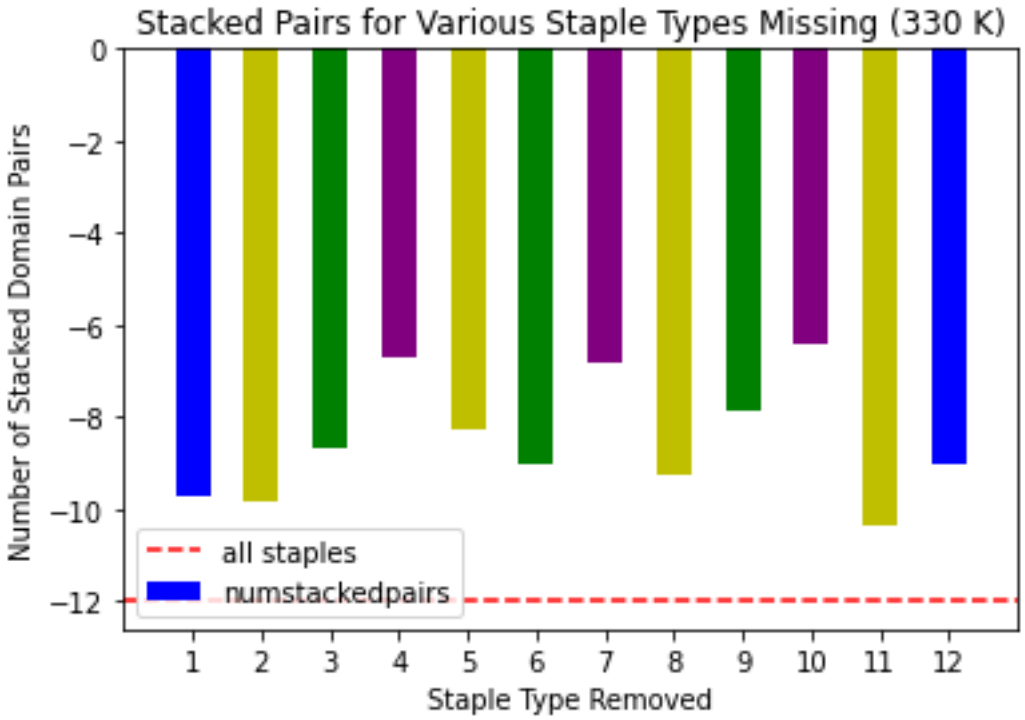


Figure 4.2: Schematic representations of the 24-binding-domain scaffold system. (a) Helical cartoon representation of the system in a fully stacked assembled configuration. (b) Representation of the system with the lattice model. The scaffold (left) and staples (right, numbered) are shown in the fully stacked assembled configuration, but for clarity have been drawn separately. A full legend for all diagram elements is provided in Figure 2.1.

It is unclear which order parameter to use for quantifying the extent of assembly. In this specific case, it was proposed by Alex to use the number of stacked domain pairs, as the fully assembled structure in a planar form should maximize the number of stacked pairs.

Results of an initial simulation are given below:

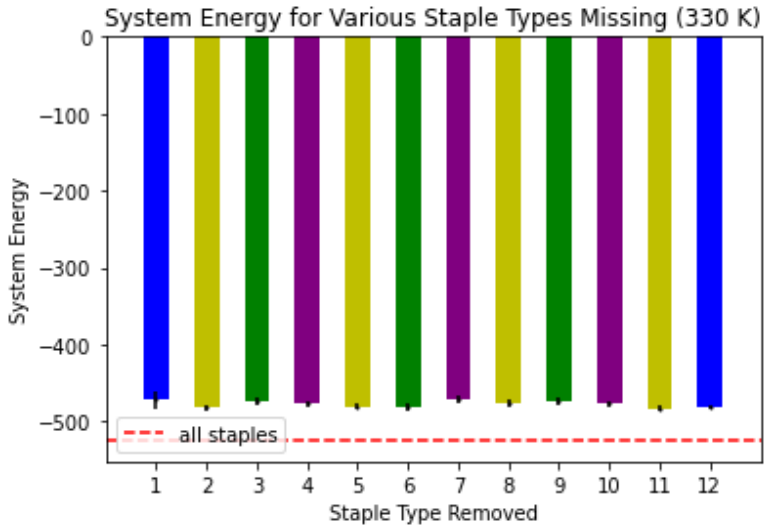


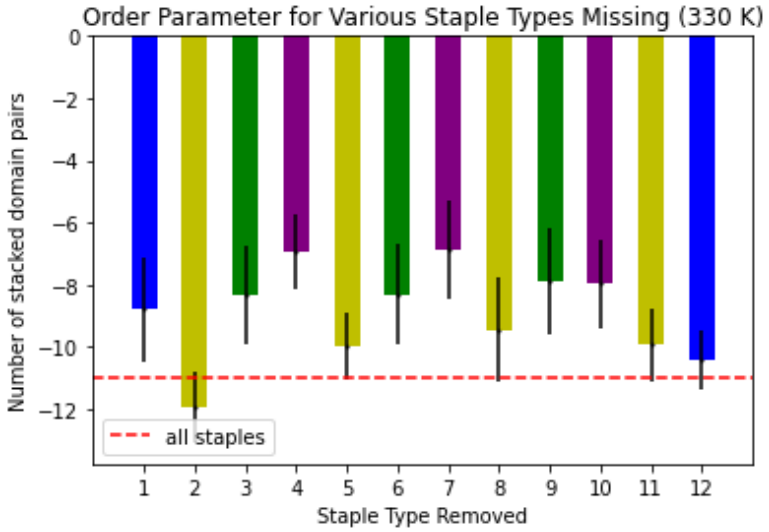
Simulation Details: 330 K, 10 000 000 MC steps, 10 000 MC step logging frequency, constant temperature, stacking energy = 1000

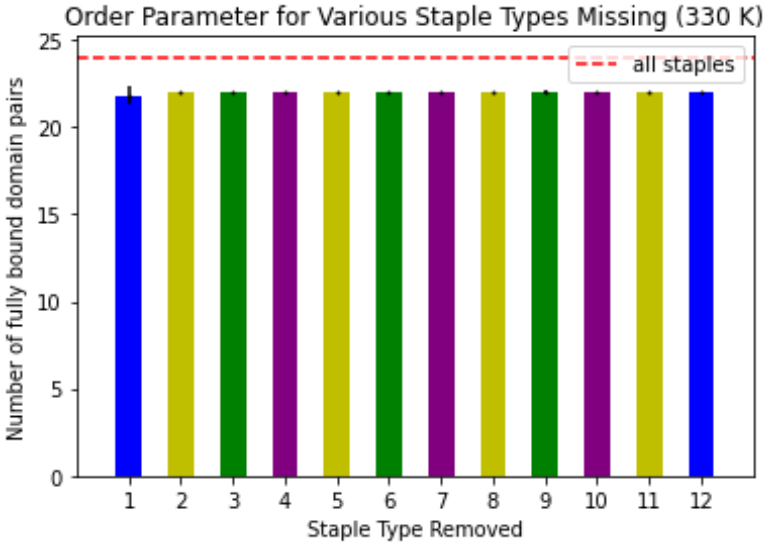
From the plot, staples 4, 7 and 10 seem to be crucial as their removal causes the greatest decrease in the number of stacked number pairs. In Alex's paper, these impose great physical restrictions, yet they are not the most restrictive, i.e. they do not have the lowest melting point. Notice that in this plot the number of stacked domain pairs for a system, where all staples are present was not actually simulated and was assumed to be 12. Moreover, the code for some reason prints out these values as negative rather than positive.

Also note that the colours of the different simulation setups are given accordingly to the colours as defined above by the categories of staple types. Although it may seem as though characteristically similar staple types have a similar effect on the order parameter studied, this is clearly not the case. The only confident distinction one can observe is for the aforementioned staples 4, 7 and 10, which clearly all show the least number of stacked domain pairs.

The simulation setup was revised. The system will all staples present was also included, as well as the stacking energy and the number of fully bound domain pairs was analysed. The results are given in the plots below:





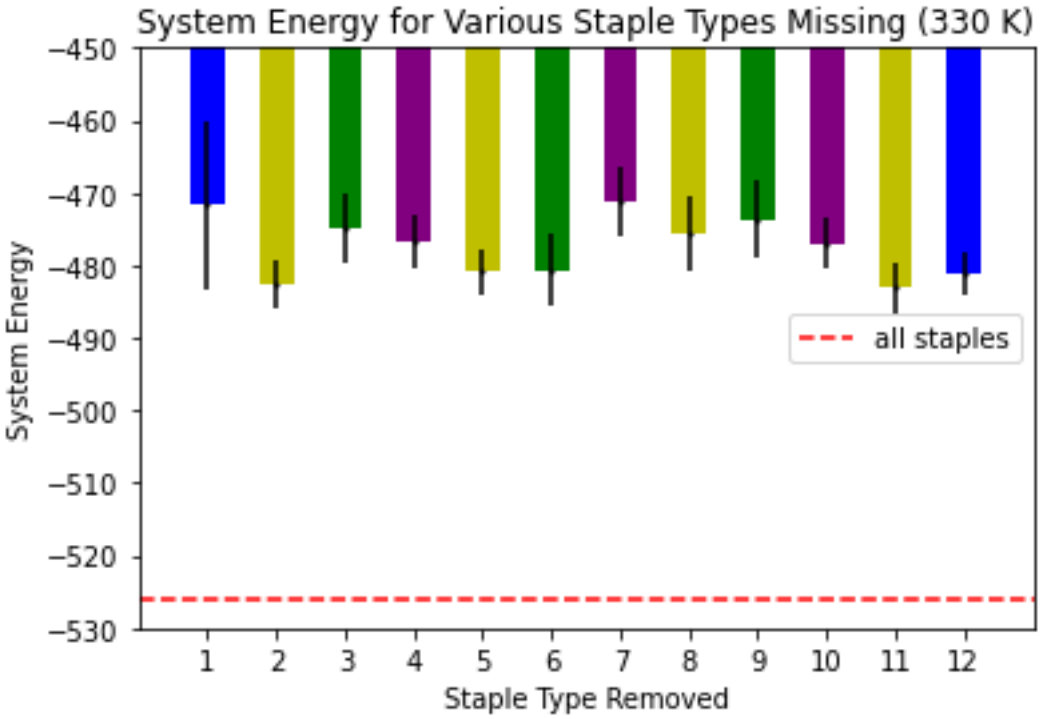


Simulation Details: 330 K, 30 000 000 MC steps, 1000 MC step logging frequency, constant temperature, stacking energy = 1000

Firstly, notice that the values for the system with all staples present is now actually represented by a simulation. Moreover, the error bars are also included. It is clear that the large uncertainty in the number of stacked domain pairs may explain why the simulation setup with staple type 2 missing shows a more stacked configuration than the system with all staples present, as this is not what we would expect to see.

The number of fully bound domain pairs does not seem to be a good indicator of the extent of the assembly to the target structure in this specific example, as this order parameter is nearly always 22 (the maximum we would expect) for all setups. Only for simulation setups with missing staple type 1, 8, 9 and 10 have a slightly lower values than 22.

Looking at the system energy, there seem to be some deviations between the various setups, but a closer look is necessary to see any clear indications:



The system energy deviations do not seem to be giving a result that would agree with the number of stacked domain pairs, hence at least one of these is not the correct way to access the extent of assembly. Nevertheless, the error bars are fairly wide and thus presumably the simulation should be run for more MC steps.

The investigation was then expanded to analyse double-removal simulation setups, for which a similar set of scripts was developed. The results are displayed in the following pages.

Simulation Details: 330 K, 10 000 000 MC steps, 1000 MC step logging frequency, constant temperature, stacking energy = 1000

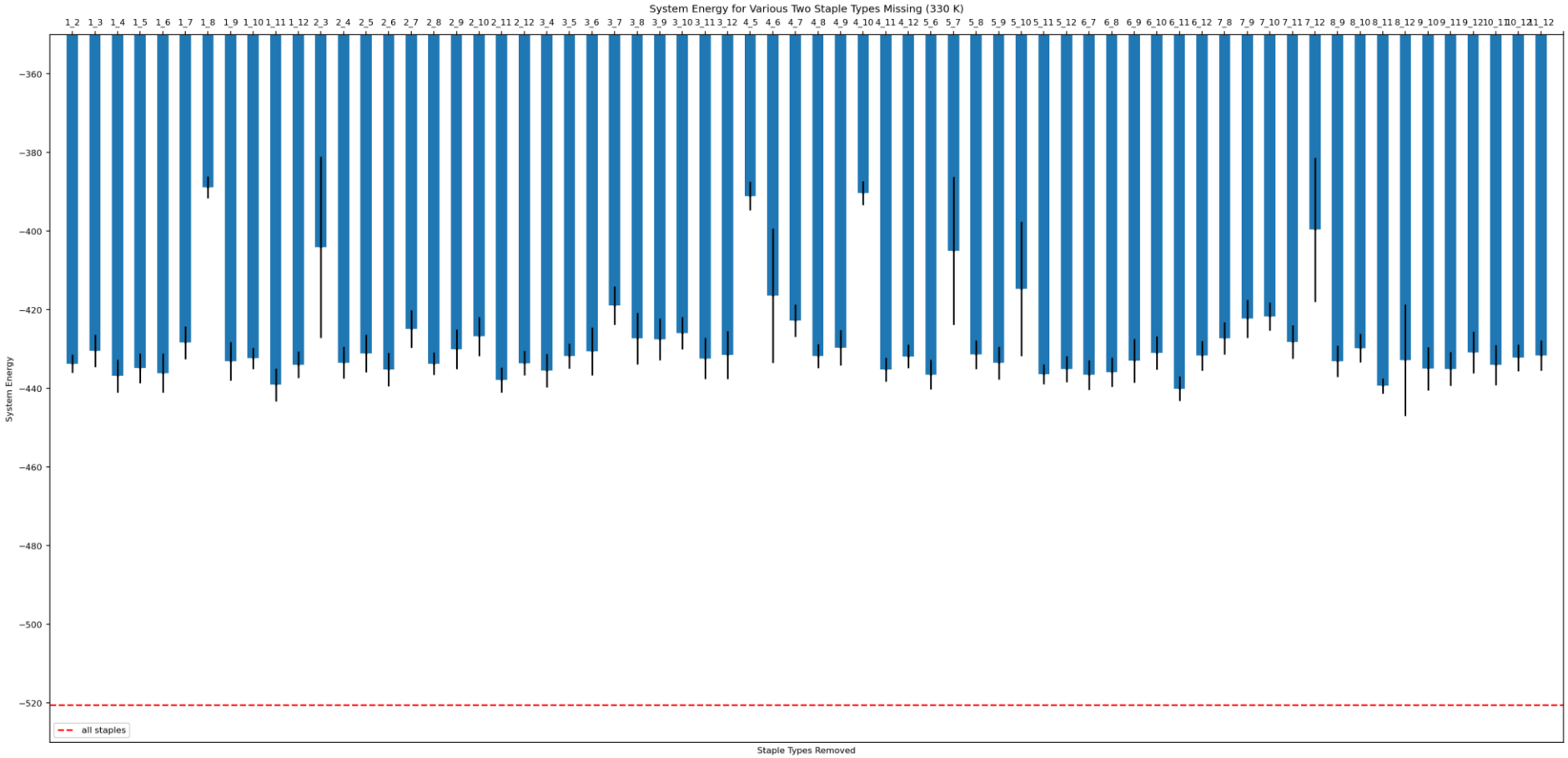
Notice that only 10 million steps were simulated for each setup, which is much less compared to the previous SR investigation. It is suggested that this should be re-run with more steps for statistically more meaningful results.

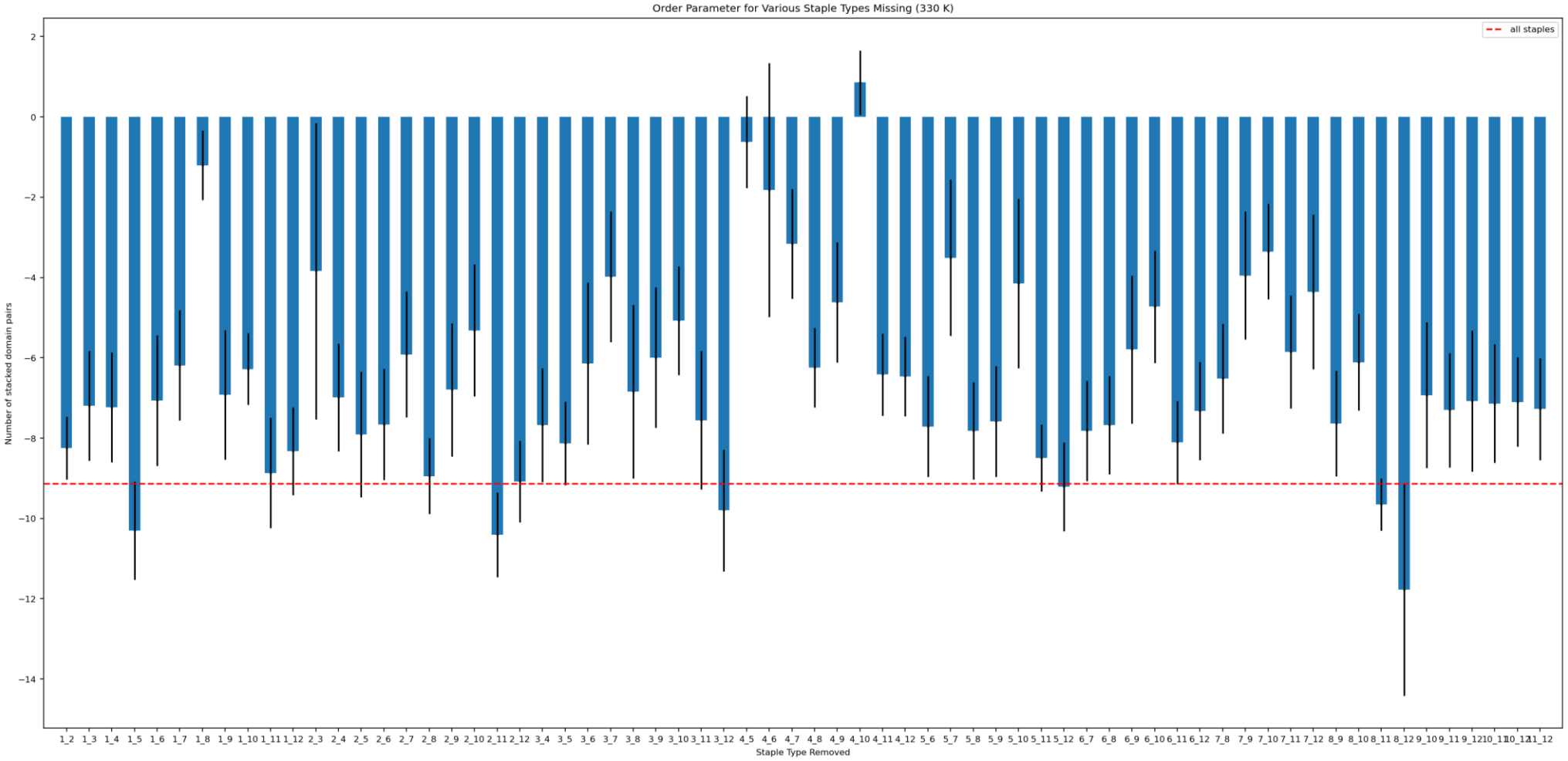
Firstly, looking at the system energy results, one may see that there are certain staple removal combinations that show a less negative energy. With some careful examination, some of these combinations such as 4 with 5 or 4 with 10, could be argued for as they form the key turning points (or sides) of the origami shape. Nevertheless, the removal of 1 with 8 or 7 with 12, should not be expected to have such a different energy, as staples 1 and 12 have both domains on the same chain, and thus should not be that important for the geometrical restrictions of assembling.

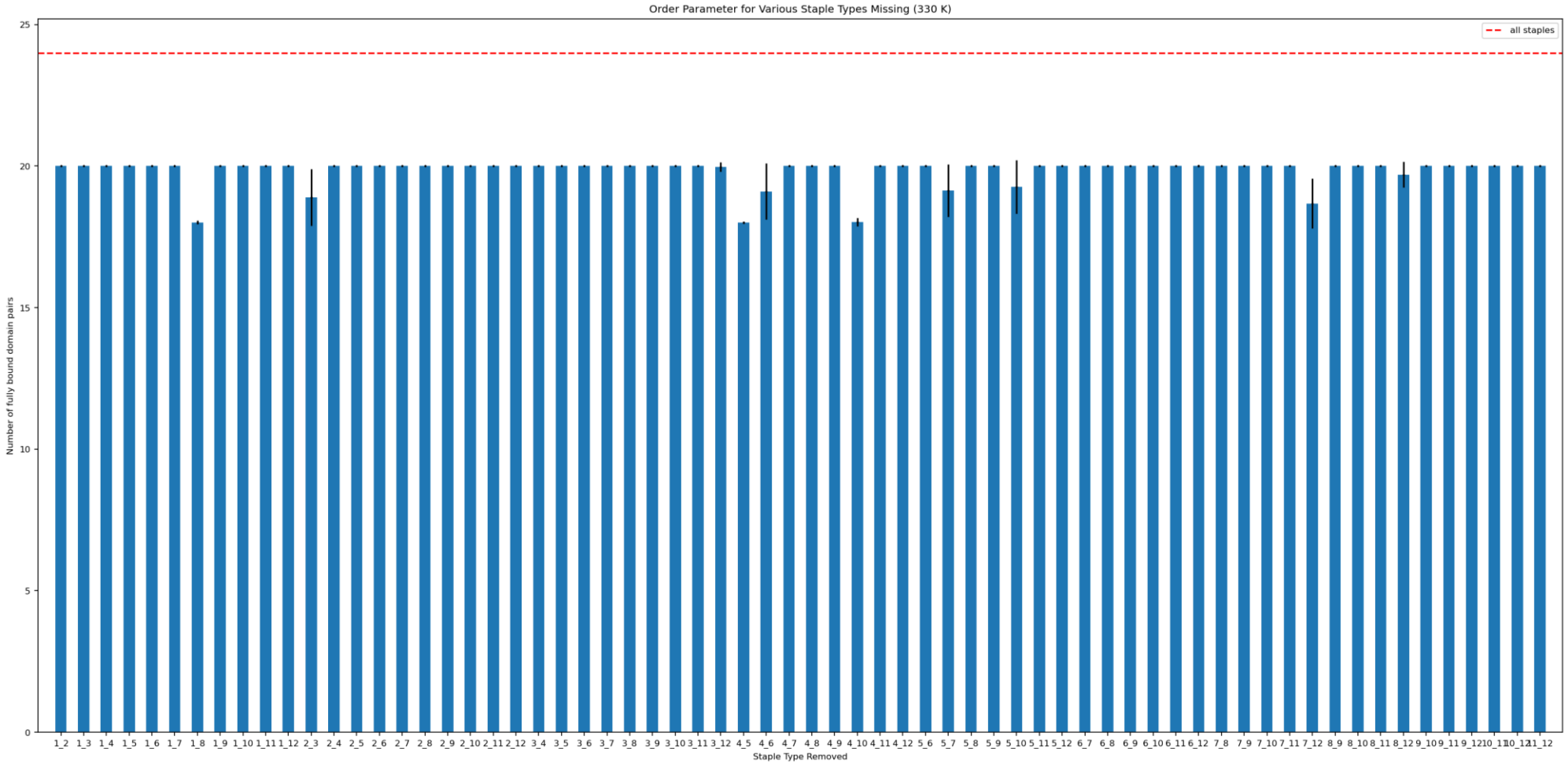
Secondly, looking at the number of stacked domain pairs, there are huge fluctuations, hence it is probably wise to not comment on these. The thing that only needs to be stressed again is therefore, that advanced sampling methods should be employed to obtain meaningful results.

Lastly, looking at the number of fully bound domain pairs, we once again see that this order parameter correctly mirrors the trends seen from the stacking energy, as explained above. Therefore, notice that removing staple 4 with staple 10 or staple 5 causes some other staple to not be able to bind, as on average the number of fully bound domain pairs is 18 rather than 20. Two investigation approaches could follow from this. Initially, it should be checked with better sampling and more MC steps, that this result is accurate, and we have not been stuck in a local energy minimum during the simulation. Afterwards, it should be checked what staple type was missing, which can be retrieved from the simulation output files.

Looking at both 4/10 and 4/5 combination, in both the staple 6 is completely missing from the final configuration (at least on average). Then staple 7 is the one that appears twice in the system for both simulation setups. Now again some important things must be stressed. Firstly, we are not that confident that these are meaningful results. Secondly, although it may come to one’s mind that staple 7 may be quite similar in the gene sequence in this specific simulation setup, and thus take over the spot of staple 6, this should not be the case. Energetically, it would still be favourable to exchange staple 7 for staple 6, hence something else must be in play. Furthermore, it is suggested that the it should be analysed where the staple 7 is actually bound to the system – whether both of the domains are (mis)bound or not, and if so, whether they still do not allow for some geometrical restriction and thus contribute at least slightly to some assembly.


System Energy for Various Two Staple Types Missing (330 K)
1_2 1_3 1_4 1_5 1_6 1_7 1_8 1_9 1_10 1_11 1_12 2_3 2_4 2_5 2_6 2_7 2_8 2_9 2_10 2_11 2_12 3_4 3_5 3_6 3_7 3_8 3_9 3_10 3_11 3_12 4_5 4_6 4_7 4_8 4_9 4_10 4_11 4_12 5_6 5_7 5_8 5_9 5_10 5_11 5_12 6_7 6_8 6_9 6_10 6_11 6_12 7_8 7_9 7_10 7_11 7_12 8_9 8_10 8_11 8_12 9_10 9_11 9_12 10_11 10_12 11_12
−360
−380
−400
−420
−440
−460
−480
−500
−520
System Energy
all staples
Staple Types Removed


Order Parameter for Various Staple Types Missing (330 K)
all staples
Number of stacked domain pairs
Staple Type Removed
2
0
−2
−4
−6
−8
−10
−12
−14
1_2 1_3 1_4 1_5 1_6 1_7 1_8 1_9 1_10 1_11 1_12 2_3 2_4 2_5 2_6 2_7 2_8 2_9 2_10 2_11 2_12 3_4 3_5 3_6 3_7 3_8 3_9 3_10 3_11 3_12 4_5 4_6 4_7 4_8 4_9 4_10 4_11 4_12 5_6 5_7 5_8 5_9 5_10 5_11 5_12 6_7 6_8 6_9 6_10 6_11 6_12 7_8 7_9 7_10 7_11 7_12 8_9 8_10 8_11 8_12 9_10 9_11 9_12 10_11 10_12 11_12


Order Parameter for Various Staple Types Missing (330 K)
all staples
Number of fully bound domain pairs
Staple Type Removed
0
5
10
15
20
25
1_2 1_3 1_4 1_5 1_6 1_7 1_8 1_9 1_10 1_11 1_12 2_3 2_4 2_5 2_6 2_7 2_8 2_9 2_10 2_11 2_12 3_4 3_5 3_6 3_7 3_8 3_9 3_10 3_11 3_12 4_5 4_6 4_7 4_8 4_9 4_10 4_11 4_12 5_6 5_7 5_8 5_9 5_10 5_11 5_12 6_7 6_8 6_9 6_10 6_11 6_12 7_8 7_9 7_10 7_11 7_12 8_9 8_10 8_11 8_12 9_10 9_11 9_12 10_11 10_12 11_12

To continue, the method of the mean staple occupancy used by Alex would be ideal to use in this investigation as well for all the different simulation setups. Moreover, some development of the bias functions or the use of umbrella sampling may be useful to improve convergence.