NANO MICRO small

Supporting Information

for *Small*, DOI: 10.1002/smll.201303442

DNA Origami Structures Directly Assembled from Intact Bacteriophages

Philipp C. Nickels, Yonggang Ke, Ralf Jungmann, David M. Smith, Marc Leichsenring, William M. Shih, Tim Liedl, and Björn Högberg**

Supplementary Information

DNA Origami Structures Assembled Directly from Intact Bacteriophages

Philipp C. Nickels[1], Yonggang Ke[2], Ralf Jungmann[2], David M. Smith[1], Marc Leichsenring[1], William M. Shih[2], Tim Liedl[1,*], Björn Högberg[3,*]

[1] Center for Nanoscience and Department of Physics, LMU Munich (Germany)

[2] Dana-Farber Cancer Institute, Harvard Medical School, Boston (USA)

[3] Department of Neuroscience, Swedish Medical Nanoscience Center, Karolinska Institute, Stockholm (Sweden)

[*] To whom correspondence should be addressed: tim.liedl@physik.lmu.de, bjorn.hogberg@ki.se

Supplementary Note S1: Folding from Bacteriophage M13

Note S1.1: Preparation of M13 Phage

The 3 different M13 bacteriophages (p7249 phage, p7560 phage, and p8064 phage) were prepared as previously described.[1] Instead of purifying the single-stranded M13 DNA we stored the PEG precipitated phage particles (re-suspended in 10mM Tris, pH=8.5) at -20°C until further use as scaffold material. Figure S1a shows a transmission electron micrograph of a purified M13 phage particle. Since phages consist of a protein shell and the genomic DNA inside, a quantitative measurement of the concentration of the phage particles via absorption spectroscopy is difficult. Titering the purified phage would give us the amount of infectious phage particles instead of the overall concentration. Thus we estimated the phage concentration via comparison to a standard DNA marker. We compared the fluorescence intensity of the target phage band on agarose gels to the intensity of a standard DNA marker with known mass value (shown in figure S1b, the double stranded 1.5kbp DNA from a 1kb DNA ladder (New England Biolabs)). The M13 phage samples were denatured prior to loading on the gel using 0.1% SDS, 10mM EDTA and incubation at 65°C for 5min.

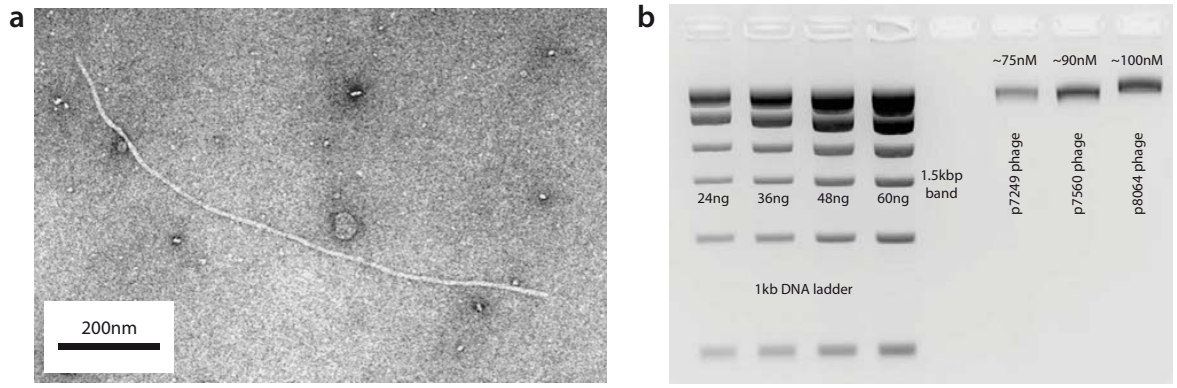


Figure S1: (a) Transmission electron micrograph of a ~1µm long filamentous M13 phage particle. **(b)** 2% agarose gel containing 0.5xTBE and 0.5µg/mL ethidium bromide run for 3.5 hours at 70V. We used 4 different concentrations of the 1kb DNA ladder. For each concentration we measured the intensity of the 1.5kbp band and together with the known mass value we derived an intensity-mass plot. Based on this intensity-mass plot we estimated the concentration of the phage particles using the intensity of the target phage bands.

Note S1.2: Assembling Structures from M13 Phage Particles

Assembly of M13 DNA origami structures was accomplished in a one-pot reaction. 5nM bacteriophage M13 particles were mixed with 100nM of every oligonucleotide staple strand (high purity salt free, Eurofins MWG Operon) in 1xTE buffer (10mM Tris-HCl pH=7.6, 1mM EDTA) containing 5mM NaCl, 0.1% SDS, 1mg/mL Proteinase K (New England Biolabs), and varying concentrations of $MgCl_2$ (12mM for the 2D rectangle, 16mM for the 6HB and the 24HB, 18mM for the three layer block). For the control samples 5nM of purified scaffold was used instead of the phage particles. The mixture was subjected to a thermal annealing ramp (using a PTC-225 DNA Engine Tetrad, MJ Research) as shown in figure S2a. It was heated to 65°C for 5 min and subsequently cooled to 25°C via a non-linear temperature ramp over the course of 2 hours for the 2D rectangle, 16 hours for the 6HB and the three layer block, and 40 hours for the 24HB.

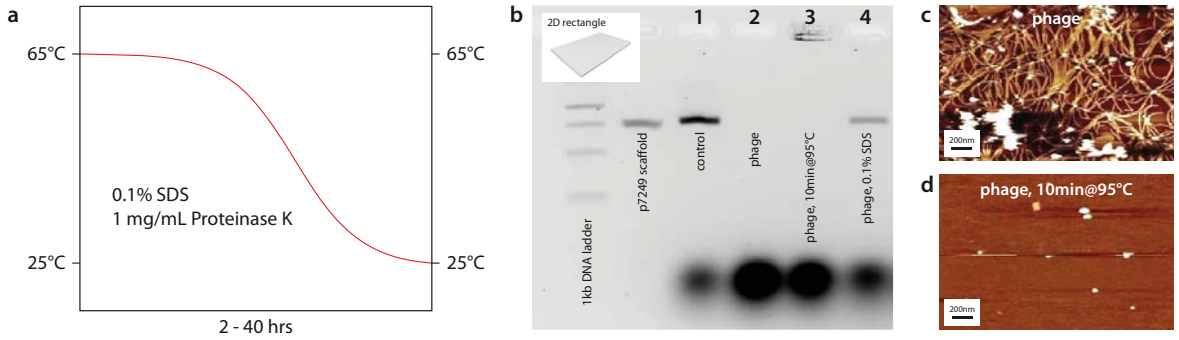


Figure S2: (a) Thermal annealing ramp used to assemble the M13 DNA origami structures (directly from phage as well as the control samples). **(b)** Comparison of different annealing procedures to fold directly from phage: only the addition of SDS results in a band of correctly assembled origami structures**. (c)** AFM image of the sample used for lane 2. **(d)** AFM image of the sample used for lane 3.

To make sure that the phage particles are denatured and origami structures are assembled from the released genomic DNA we tested different annealing schemes. The samples were compared to a control sample conventionally folded from purified scaffold (figure S2b, lane 1). Substituting the purified scaffold with phage particles without any denaturing agent or treatment followed by the depicted annealing ramp (figure S2a) did not yield a band of folded structures (lane 2). The AFM image of such a sample in figure S2c shows that the phages are still intact. A 10 min heat denaturing step followed by the described annealing

did not result in a detectable band of folded products as well (lane 3). The AFM image of such a sample in figure S2d shows that the phage particles are denatured but only a few origami structures are formed. The addition of SDS as denaturing agent without any extra heat denaturing steps yields a band of correctly folded structures migrating at the same speed as the control band (lane 4). Proteinase K is not needed for the folding process itself. However, it digests the M13 capsid protein debris and thus results in a cleaner background in AFM/TEM imaging.

Note S1.3: Assembling Structures from M13 infected liquid *E.coli* Culture

M13 bacteriophage was amplified in an *E.coli* liquid culture as previously described.[1] At the point of harvest the *E.coli* cells were pelleted via centrifugation at 3,000rcf, 4°C for 30min. The cell pellet was discarded and the supernatant containing the unpurified bacteriophage was kept at 4°C until further usage as scaffold material. The concentration of bacteriophage particles in the crude suspension was estimated via gel electrophoresis as described in note S1.1 and figure S1. A typical liquid culture M13 prep yielded a bacteriophage concentration of about 1 to 2nM in the crude suspension after *E.coli* cell removal.

We tested whether a crowded environment full of biological compounds might inhibit the self-assembly process via the addition of fetal bovine serum (FBS). Figure S3a shows the agarose gel analysis data of the assembly of the M13 based 2D rectangle. We added four different concentrations between 10 and 40 volume-% FBS to the assembly reaction. The FBS was heat-inactivated at 65°C for 30 minutes prior to mixing the components to prevent DNA degradation while preparing the sample. A decrease in assembly yield with increasing FBS could not be observed.

The assembly of the M13 DNA origami structures from the crude bacteriophage suspension was accomplished in a one-pot reaction. About 1 to 1.5nM M13 bacteriophage (¾ of the final sample volume was crude M13 suspension) was mixed with 12.5nM of every oligonucleotide staple strand for the six helix bundle structure (high purity salt free, Eurofins MWG Operon) in 1xTE buffer (10mM Tris-HCl pH=7.6, 1mM EDTA) containing a final concentration of 16mM $MgCl_2$, 62mM NaCl, 0.1% SDS and 1mg/mL Proteinase K (New England Biolabs). Note: the growth medium used for the M13 prep contains 83mM NaCl,

thus the high NaCl concentration in the final sample. Since it also contains 5mM $MgCl_2$, we adjusted the amount of added $MgCl_2$ to 12.25mM resulting in the final concentration of 16mM. The sample was subjected to the thermal annealing procedure described in note S1.2 (16 hour total annealing time for the 6 helix bundle).

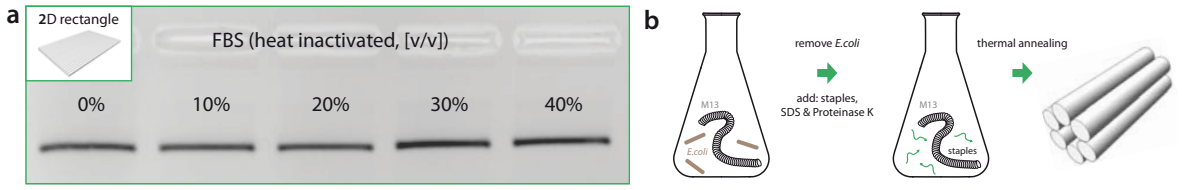


Figure S3: (a) Assembly of the 2D rectangle from purified M13 phage with different amounts of FBS. The crowded environment does not affect the assembly yield. The only detectable difference is the decrease in migration speed with increasing FBS concentrations due to the increasing amounts of sodium ions added with the FBS. **(b)** Illustration of the assembly process directly from an M13 infected *E.coli* liquid culture: first we removed the bacterial cells via centrifugation. Next, all components needed for the assembly are added and the structure is folded via thermal annealing without any purification of the phage particles.

Note S1.4: Agarose Gel Electrophoresis and Gel Purification of M13 DNA Origami Structures

Folded M13 origami constructs were electrophoresed on 2% agarose gels containing 0.5xTBE buffer (45mM Tris, 45mM boric acid, 1mM EDTA), 11mM $MgCl_2$, and 0.5 µg/mL ethidium bromide for 3 hours at 5.5 V/cm cooled in an ice water bath. Bands were visualized with ultraviolet light and physically extracted. DNA was recovered by pestle-crushing excised bands, freezing for 5min followed by centrifugation for 10min at 5000rcf in a microfuge at 4°C using Freeze'N'Squeeze DNA Gel Extraction spin columns (Bio-Rad). Recovered material in the flow-through was stored at -20°C for further usage. The 1kb DNA ladder was purchased from New England Biolabs.

Note S1.5: Transmission Electron Microscopy of M13 DNA Origami Structures

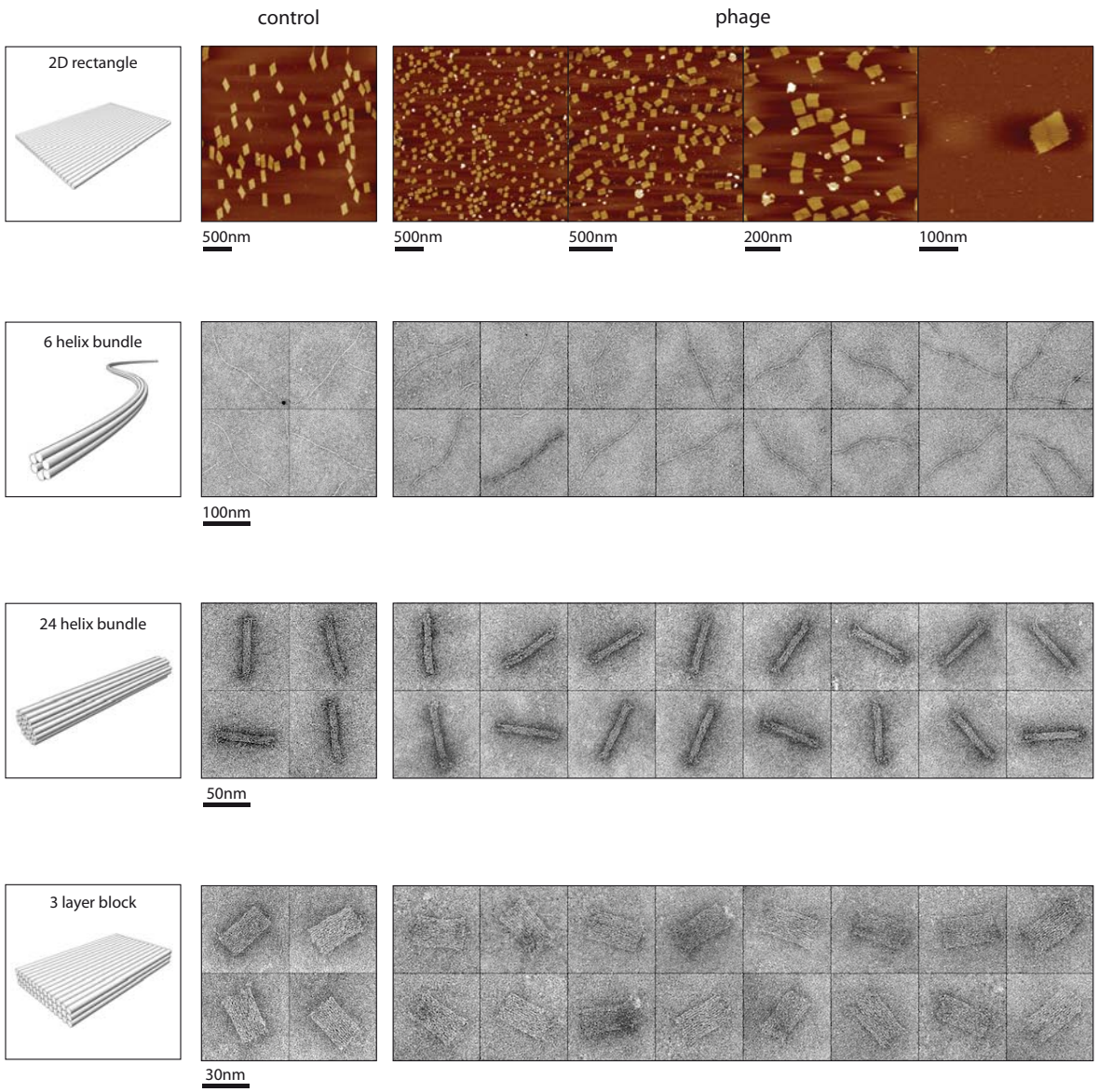
For TEM imaging, 3µL of the gel purified M13 origami solution was adsorbed onto glow-discharged formvar/carbon-coated TEM grids (Plano) and then stained using a 2% aqueous uranyl formate solution containing 25mM sodium hydroxide. Imaging was performed using

a JEM1011 transmission electron microscope (JEOL) operated at 100 kV equipped with a FastScan-F114 camera (TVIPS).

Note S1.6: Atomic Force Microscopy of M13 DNA Origami Structures

For AFM imaging, the folded M13 DNA origami structures (the 2D rectangle) were dialyzed against 500mL 1xTE buffer (10mM Tris-HCl pH=7.6, 1mM EDTA) containing 12mM $MgCl_2$ for 2 hours at room temperature in 3.5kDa MWCO dialysis units (Slide-A-Lyzer MINI Dialysis Unit 3.5kDa MWCO, Thermo scientific) to remove the SDS. Samples were imaged in tapping mode using a Multimode III AFM (Veeco Metrology Group, now Bruker AXS). Imaging was performed in 1xTAE buffer solution containing 12mM $MgCl_2$ with SNL-10 sharp nitride cantilevers (Veeco, now Brucker AFM Probes) using resonance frequencies between 7-9 kHz of the 0.24N/m force constant cantilever. 25µL of buffer solution was dropped onto a freshly cleaved mica surface (Plano). 5µL of the dialyzed origami solution was added to the buffer on the mica surface. Imaging parameters were optimized for best image quality while maintaining the highest possible set point to minimize damage to the samples. Images were post-processed by subtracting a 1st order polynomial from each scan line.

Figure S4: Additional AFM and TEM Data of M13 DNA Origami Structures



Supplementary Note S2: Folding from Bacteriophage λ

Note S2.1: Preparation of λ Phages

Bacteriophage λ particles were prepared as follows: wild-type λ-DNA was packaged using a commercially available packaging extract (MaxPlax™ Lambda Packaging Extract, EPICENTRE© Biotechnologies) following the manufacturers instructions. This packaging reactions yielded a packaging efficiency of > 10^9pfu/µg of the wild-type λ-DNA. A high titer stock of λ phages for long term storage was prepared via plate lysis and elution. DMSO was added to the phage suspension (final concentration of 7% (v/v)), aliquots were plunged in liquid nitrogen and afterwards stored at -80°C.

Phage particles for folding experiments were prepared via a small scale liquid culture. A 50mL LB culture (supplemented with 0.2% (v/v) maltose and 10mM $MgSO_4$) of *E.coli* XL-1 Blue cells (Agilent Technologies) was incubated at 30°C with shaking overnight (the low temperature ensures that the cells will not overgrow: phages can adhere to nonviable cells resulting in a decreased titer of the prep). The bacterial cells were pelleted via centrifugation (600rcf for 10min at room temperature) and the supernatant was discarded. The cells were re-suspended and diluted to an OD_{600} of 0.5 with 10mM $MgSO_4$. 100µL of the prepared cells were mixed with 1x10^6pfu (100µL of a 10^7pfu/mL dilution of the high titer long term stock) and incubated at 37°C for 20min. 4mL of pre-warmed LB medium was added followed by incubation at 37°C with vigorous shaking (300rpm). After 5 hours, one drop of chloroform was added (about 50µL) and the suspension was incubated for another 15min at 37°C with shaking. The debris of the lysed bacterial cells was removed via centrifugation at 13,000rcf, 4°C for 15min. DNaseI was added to a final concentration of 20µg/mL, RNaseA to a final concentration of 8 µg/mL, $CaCl_2$ to a final concentration of 10mM. The suspension was incubated at 37°C for 30min. Afterwards the suspension was filtered using a sterile 0.45µm syringe filter. The phage particles were precipitated (using 10% PEG-8000, 1M NaCl and incubation in ice water for 90min) and subsequently pelleted via centrifugation at 16,000rcf, 4°C for 20min. The pelleted phage was re-suspended in SM buffer (100mM NaCl, 8mM $MgSO_4$, 50mM Tris-HCl pH 7.6, autoclaved). PEG was removed via addition of an equal volume of chloroform followed by slow vortexing and

centrifugation at 3,000rcf, 4°C for 10min. The purified phage suspension was stored at 4°C until further use as scaffold material. [2]

Figure S5a shows a transmission electron micrograph of a purified λ phage particle. Figure S5b shows an agarose gel assay of the different steps of the phage purification procedure after the small scale liquid lysate prep. The concentration of the purified phages was estimated as described in Note S1.1 for the M13 phages. We compared the fluorescence intensity of the purified λ phage band to the intensity of the standard DNA marker with known mass value.

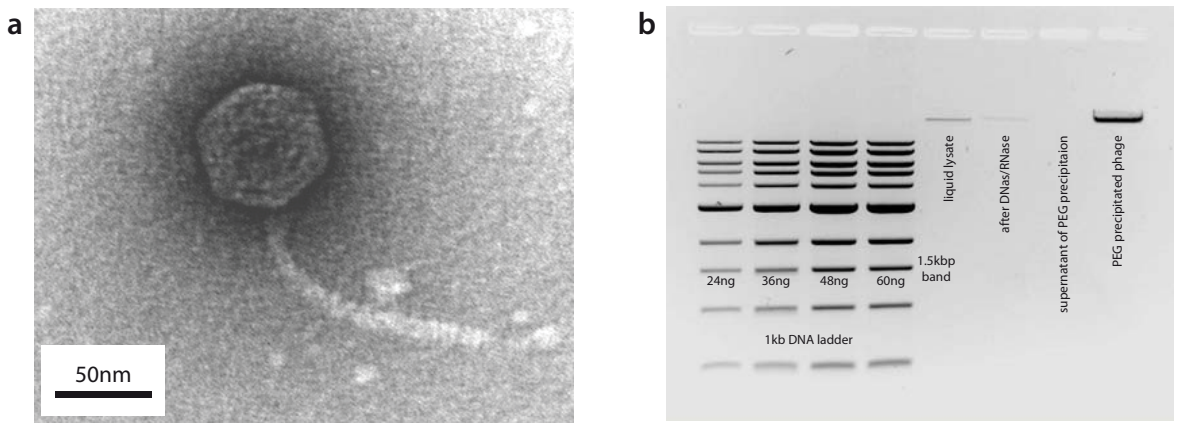


Figure S5: (a) Transmission electron micrograph of a purified λ phage particle. **(b)** 0.7% agarose gel containing 0.5xTBE and 0.5µg/mL ethidium bromide run for 3 hours at 70V. We used 4 different concentrations of the 1kb DNA ladder. For each concentration we measured the intensity of the 1.5kbp band and together with the known mass value we derived an intensity-mass plot. Based on this intensity-mass plot we estimated the concentration of the phage particles using the intensity of the PEG precipitated λ phage band.

Note S2.2: Denaturing Agarose Gel Electrophoresis of λ-DNA and Phages

Native and denatured λ-DNA (commercially available from New England Biolabs (NEB) as well as our own) and λ phage particles were compared on a 0.7% agarose gel containing 1xTAE buffer (40mM Tris, 40mM acetic acid, 1mM EDTA) containing 1M urea and 0.5µg/mL ethidium bromide. Denatured samples contained 8M urea and were incubated at 80°C for 5min prior to loading.[3] The gel was run for 3 hours at 5.5V/cm cooled in an ice water bath and bands were visualized with ultraviolet light (image shown in figure S6).

The native NEB λ-DNA results in a nice single band (lane 1). The denatured NEB λ-DNA smears over a range of more than 10kbp (lane 2). This indicates the high amount of nicks in the commercially available λ-DNA molecules, most likely due to mechanical degradation from pipetting and storage. We ligated the NEB λ-DNA with *E.coli* DNA Ligase (New England Biolabs) to repair the backbone (4 hours incubation at 16°C followed by heat inactivation at 65°C for 20min). To prevent the creation of long concatamers we added and thermally annealed a 5-T overhang oligo complementary to one of the cohesive ends to the λ-DNA. Ligation was partly successful: less smearing was observed as well as a high amount of material not entering the gel (lane 3). This indicates that the strategy to prevent concatamers did not work sufficiently. Our own purified λ-DNA (lane 4) as well as the denatured phage particles (lane 5) show a nice single-stranded band. Note that there is a band migrating at the same speed as the native λ-DNA occurring in most denatured samples. This might indicate that not 100% of the dsDNA molecules denature via the 5min incubation step or a fraction of the DNA re-anneals afterwards.

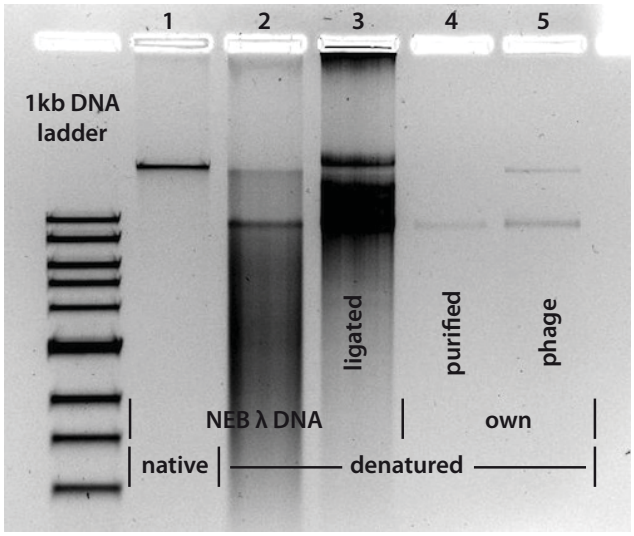


Figure S6: 0.7% agarose denaturing gel containing 1M urea. **Lane 1:** native NEB λ- DNA. **Lane 2:** denatured NEB λ-DNA. **Lane 3:** denatured NEB λ-DNA after ligation. **Lane 4**: denatured purified homemade λ-DNA. **Lane 5:** denatured λ-DNA directly from phage particles. (Denatured samples contained 8M urea and were heated to 65°C for 5min prior to loading on to the gel.)

Note S2.3: λ-DNA Melting Curves

We measured the melting behavior of λ-DNA in the presence of six different formamide concentrations (figure S7). Each sample contained 5µg of λ-DNA (Invitrogen), 1xTE buffer (10mM Tris-HCl pH=7.6, 1mM EDTA), and 12mM $MgCl_2$. Due to the spectral overlay of formamide and DNA at 260nm we recorded the absorption at 270nm in a UV/Vis spectrophotometer (JASCO V-650). The samples were heated to 97°C, cooled to 25°C and again heated to 97°C in intervals of 0.1°C at a rate of 0.5°C per min. The shift of the melting temperature in the presence of formamide to lower temperatures corresponds nicely with the reported value of 0.64°C per volume percent of formamide.[4]

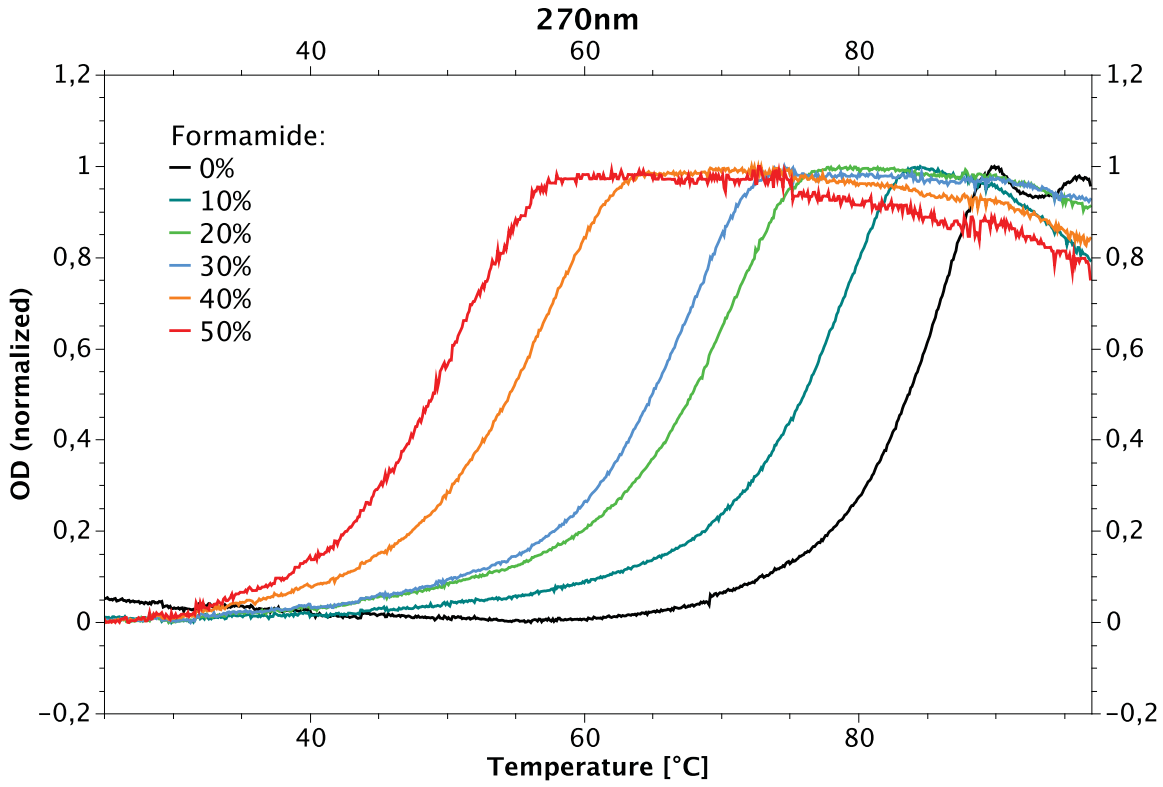


Figure S7: Melting curves of λ-DNA in the presence of different formamide concentrations. The fact that the blue curve is not showing the melting temperature as expected for 30% formamide is most likely due to a pipetting error.

Note S2.4: Assembling Structures from λ Phage Particles

Assembly of λ DNA origami structures was accomplished in a one-pot reaction. 0.2nM bacteriophage λ particles were mixed with 10nM of every oligonucleotide staple strand (reverse-phase cartridge purified, Bioneer Inc.) in 1xTE buffer (10mM Tris-HCl pH=7.6, 1mM EDTA) containing 12mM $MgCl_2$, 0.1% SDS, and 40% formamide. For the control samples between 0.2 and 2nM of commercially available λ-DNA (New England Biolabs) was used instead of the phage particles. The mixture was subjected to a thermal annealing ramp (using a PTC-225 DNA Engine Tetrad, MJ Research) as described in figure S7. The samples were dialyzed against 500mL 1xTE buffer (10mM Tris-HCl pH=7.6, 1mM EDTA) containing 12mM $MgCl_2$ at 4°C overnight in 3.5kDa MWCO dialysis units (Slide-A-Lyzer MINI Dialysis Unit 3.5kDa MWCO, Thermo scientific) with gentle mixing using a magnetic stirrer.

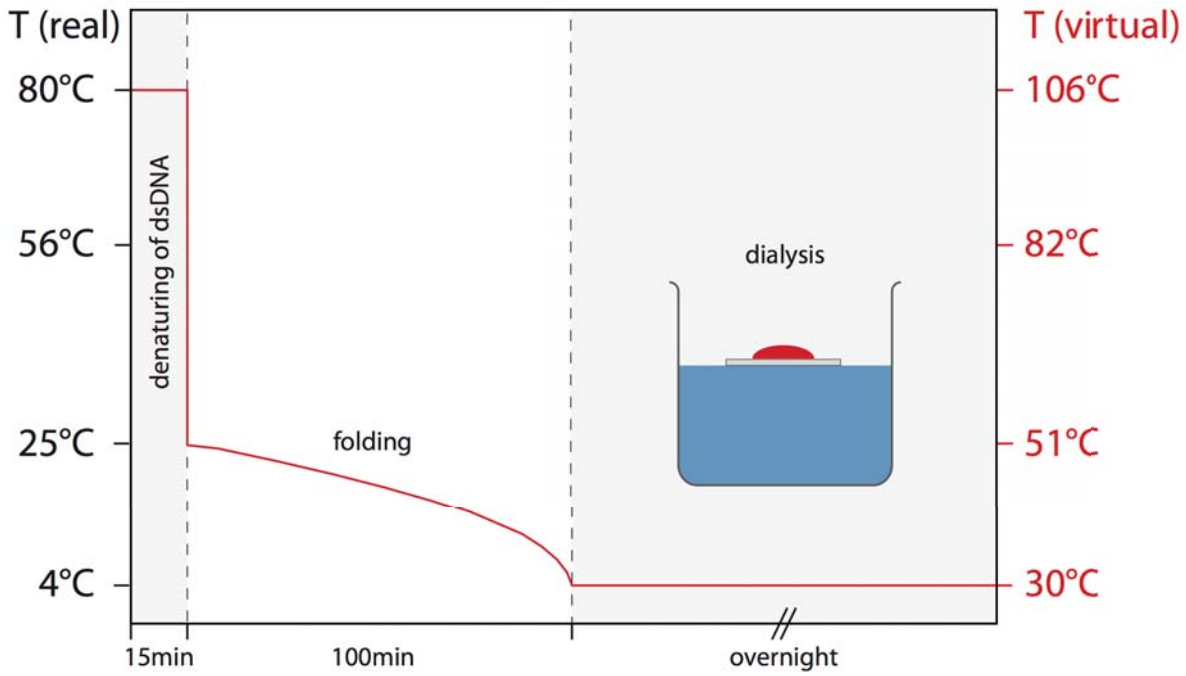


Figure S8: The sample is heated to 80°C for 15min to denature the phage particles, release the genomic DNA and melt the dsDNA. A temperature drop to 25°C in the presence of the high excess of staple strands prevents re-annealing of the dsDNA. A nonlinear temperature ramp to 4°C over the course of 100min is applied to assemble the structures. Overnight dialysis at 4°C is used to remove the formamide as well as the SDS.

Note S2.5: Agarose Gel Electrophoresis of λ DNA Origami Structures

Folded constructs were electrophoresed on 0.7% agarose gels containing 0.5xTBE buffer (45mM Tris, 45mM boric acid, 1mM EDTA), 11mM $MgCl_2$ and 0.5µg/mL ethidium bromide for 3 hours at 5.5 V/cm cooled in an ice water bath. Bands were visualized with ultraviolet light.

Note S2.6: Atomic Force Microscopy of λ DNA Origami Structures

Samples were imaged in tapping mode using a Multimode III AFM (Veeco Metrology Group, now Bruker AXS). Imaging was performed in 1xTAE buffer solution containing 12mM $MgCl_2$ with SNL-10 sharp nitride cantilevers (Veeco, now Brucker AFM Probes) using resonance frequencies between 7-9 kHz of the 0.24 N/m force constant cantilever. 5µL of the dialyzed origami solution was dropped onto a freshly cleaved mica surface (Plano). After 5min, 25µL of buffer solution was added to the sample on the mica surface. Imaging parameters were optimized for best image quality while maintaining the highest possible set point to minimize damage to the samples. Images were post-processed by subtracting a 1st order polynomial from each scan line.

Note S2.7: Transmission Electron Microscopy of λ DNA Origami Structures

For TEM imaging, 3µL of the dialyzed origami solution was adsorbed onto glow-discharged formvar/carbon-coated TEM grids (Plano) and then stained using a 2% aqueous uranyl formate solution containing 25mM sodium hydroxide. Imaging was performed using a JEM1011 transmission electron microscope (JEOL) operated at 100 kV equipped with a FastScan-F114 camera (TVIPS).

Figure S9: Additional AFM and TEM Images of λ DNA Origami Structures

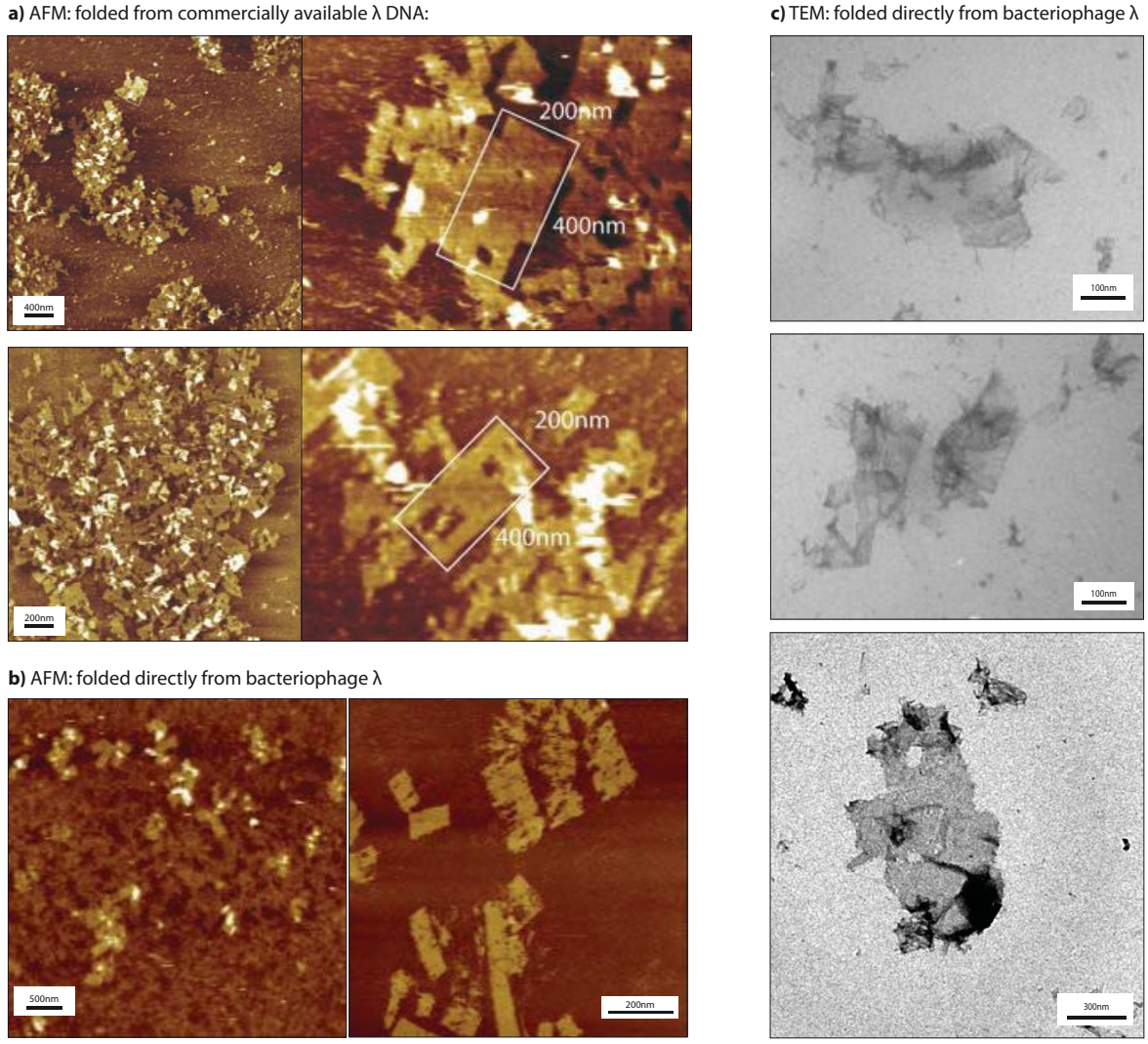


Figure S9: **(a)** AFM images of samples prepared with commercially available λ-DNA. **Left:** Structures folded with 2nM λ -DNA. (Note: the small M13 based triangular structures in the upper most left image result from a tip used before to image these triangles. However, they serve as a nice physical scale bar.) **Right:** The same sample imaged with higher magnification. **(b)** AFM data of structures folded directly from λ phage particles. The high amount of phage protein debris covering the surface is clearly visible. **(c)** Negative stain TEM images of structures folded directly from λ phage particles.

Supplementary Note S3: DNA Origami Designs and DNA Sequences

Note S3.1: Design of M13 DNA Origami Structures

All M13 based structures were designed using the software package caDNAno (version 0.23, http://www.cadnano.org/legacy).[5] The 2D rectangle was slightly modified from the original version described by Rothemund: the scaffold was permutated and the global twisting was reduced by the introduction of targeted deletions.[6,7] The 'M13' writing was realized by inserting dumbbell shaped hairpins at the positions indicated in figure S10 (see reference [6] for the hairpin sequence). The scaffold routing of the six helix bundle was as well permutated compared to the original version described by Douglas et al.[1] The 24 helix bundle as well as the three layer block were used as previously described without any changes.[8,9] The four designs are shown in figures S10-13.

Note S3.2: Design of the λ DNA Origami Structure

For each of the two single strands of the double-stranded molecule we designed a separate structure. These two structures are geometrically identical but chemically distinct rectangles, each 100x400nm in size. Each of these monomers consists of 154 adjacent antiparallel double helices of 320bp in length and is folded by an individual set of staples (3150 individual oligonucleotides of 32nt length in total). The two monomers are connected via an extra set of staples and together form the complete 200x400nm large rectangular object. The scaffold is routed through the λ DNA origami structure similar to the M13 based 2D rectangle. Staple double crossovers are implemented every 32 base pairs. A python script (appendix S1) which uses the scheme shown in figure S14 was used to draw the entire design of the λ DNA origami structure (figure S16), to save a caDNAno compatible file and compute all the staple sequences. Since the caDNAno software was not able to open the generated file (the software crashed due to the high number of bases populated), we used the script to generate a truncated design with only 12 double helices as a control. The staple sequences generated by the script matched perfectly with the sequences generated independently by caDNAno. Figure S15 shows the caDNANo design of the truncated 12 helix tall structure. (A complete list of all staple sequences used is available upon request.)

References

[1] S. M. Douglas, J. J. Chou, W. M. Shih, *Proceedings of the National Academy of Sciences* **2007**, *104*, 6644–6648.

[2] J. Sambrook, *Molecular Cloning: a Laboratory Manual, Third Edition*, Cold Spring Harbor Laboratory Press, **2001**.

[3] E. Hegedus, E. Kokai, A. Kotlyar, V. Dombradi, G. Szabo, *Nucleic Acids Research* **2009**, *37*, e112–e112.

[4] R. D. Blake, S. G. Delcourt, *Nucleic Acids Research* **1996**, *24*, 2095–2103.

[5] S. M. Douglas, A. H. Marblestone, S. Teerapittayanon, A. Vazquez, G. M. Church, W. M. Shih, *Nucleic Acids Research* **2009**, *37*, 5001–5006.

[6] P. W. K. Rothemund, *Nature* **2006**, *440*, 297–302.

[7] Y. Ke, S. Douglas, M. Liu, J. Sharma, A. Cheng, A. Leung, Y. Liu, W. Shih, H. Yan, *Journal of the American Chemical Society* **2009**, *131*, 15903–15908.

[8] A. Kuzyk, R. Schreiber, Z. Fan, G. Pardatscher, E.-M. Roller, A. Högele, F. C. Simmel, A. O. Govorov, T. Liedl, *Nature* **2012**, *483*, 311–314.

[9] I. H. Stein, V. Schüller, P. Böhm, P. Tinnefeld, T. Liedl, *Chem. Eur. J. of Chem. Phys*. **2011**, 12, 689–695.

Figure S10:

Design of the 2D Rectangle

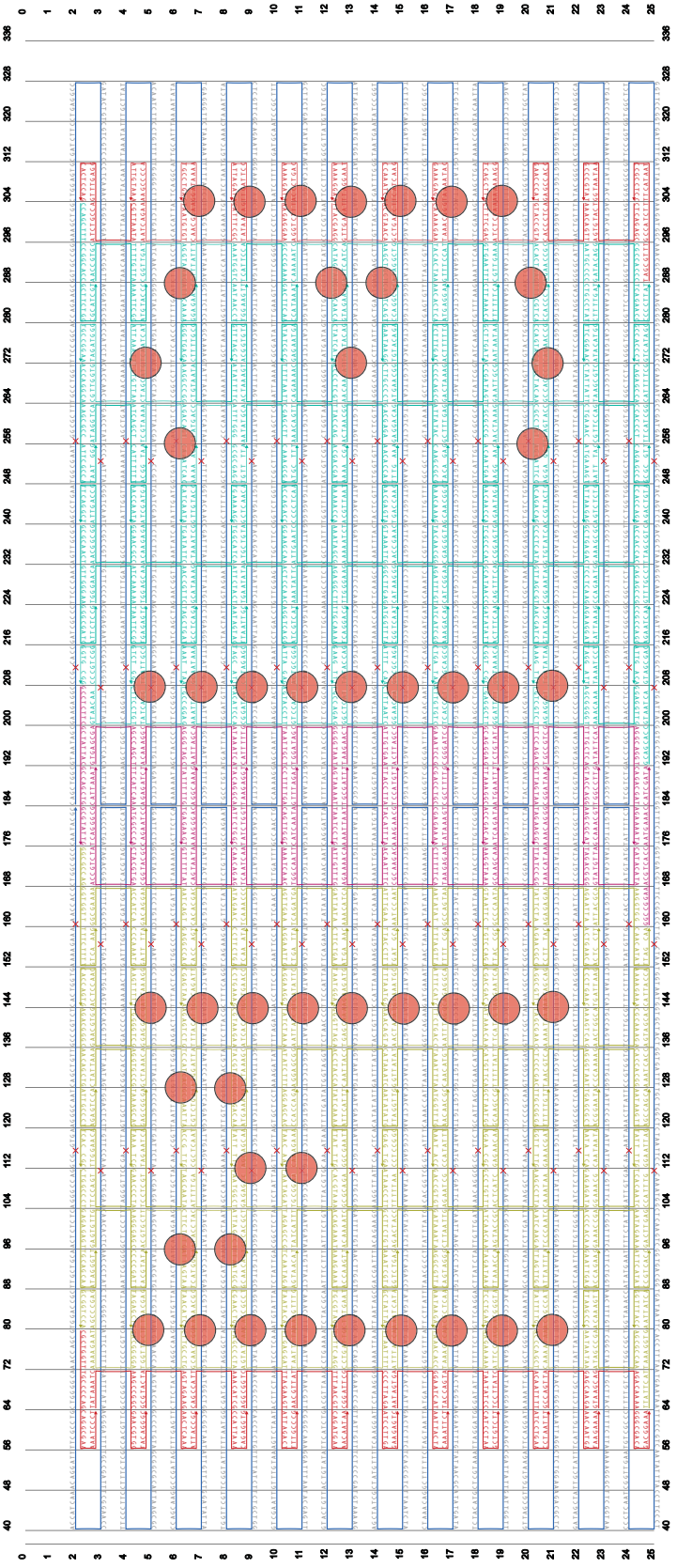


Figure S11:
Design of the 6 Helix Bundle

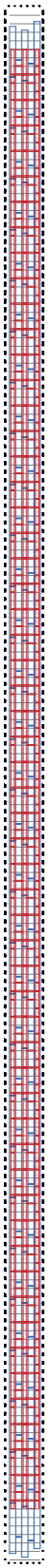
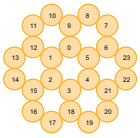


Figure S12:

Design of the 24 Helix Bundle



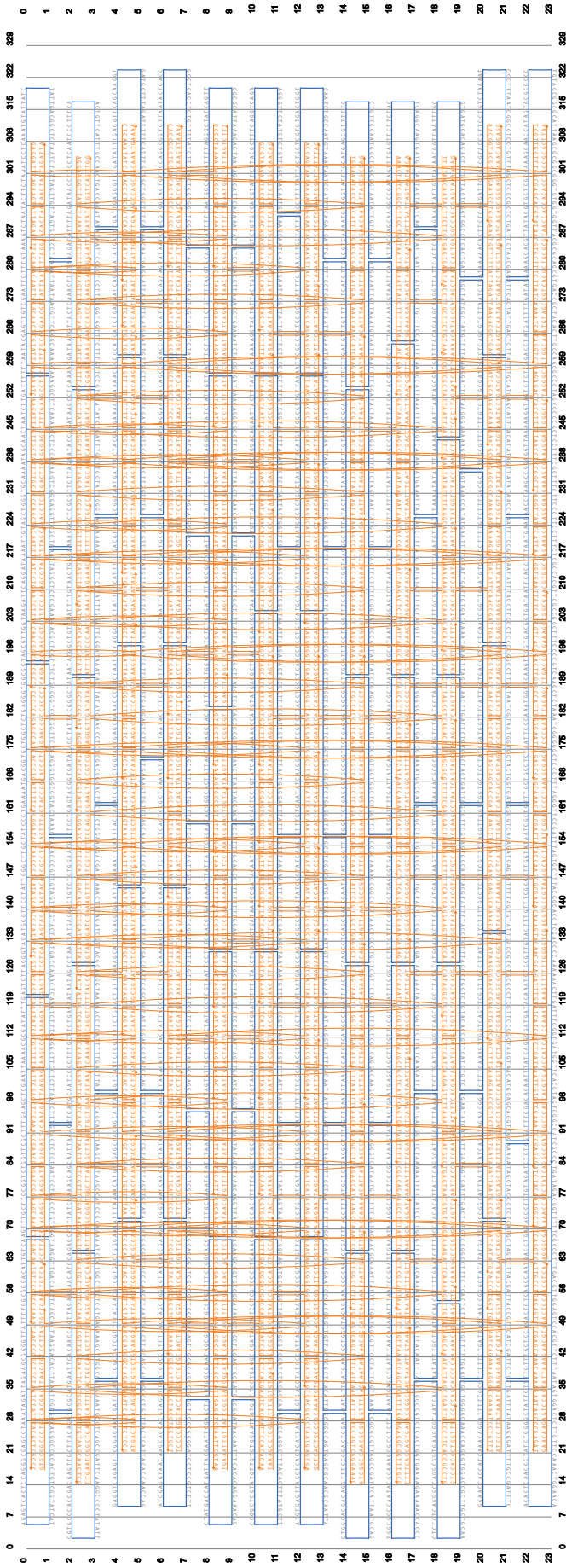


Figure S13:

Design of the 3 Layer Block

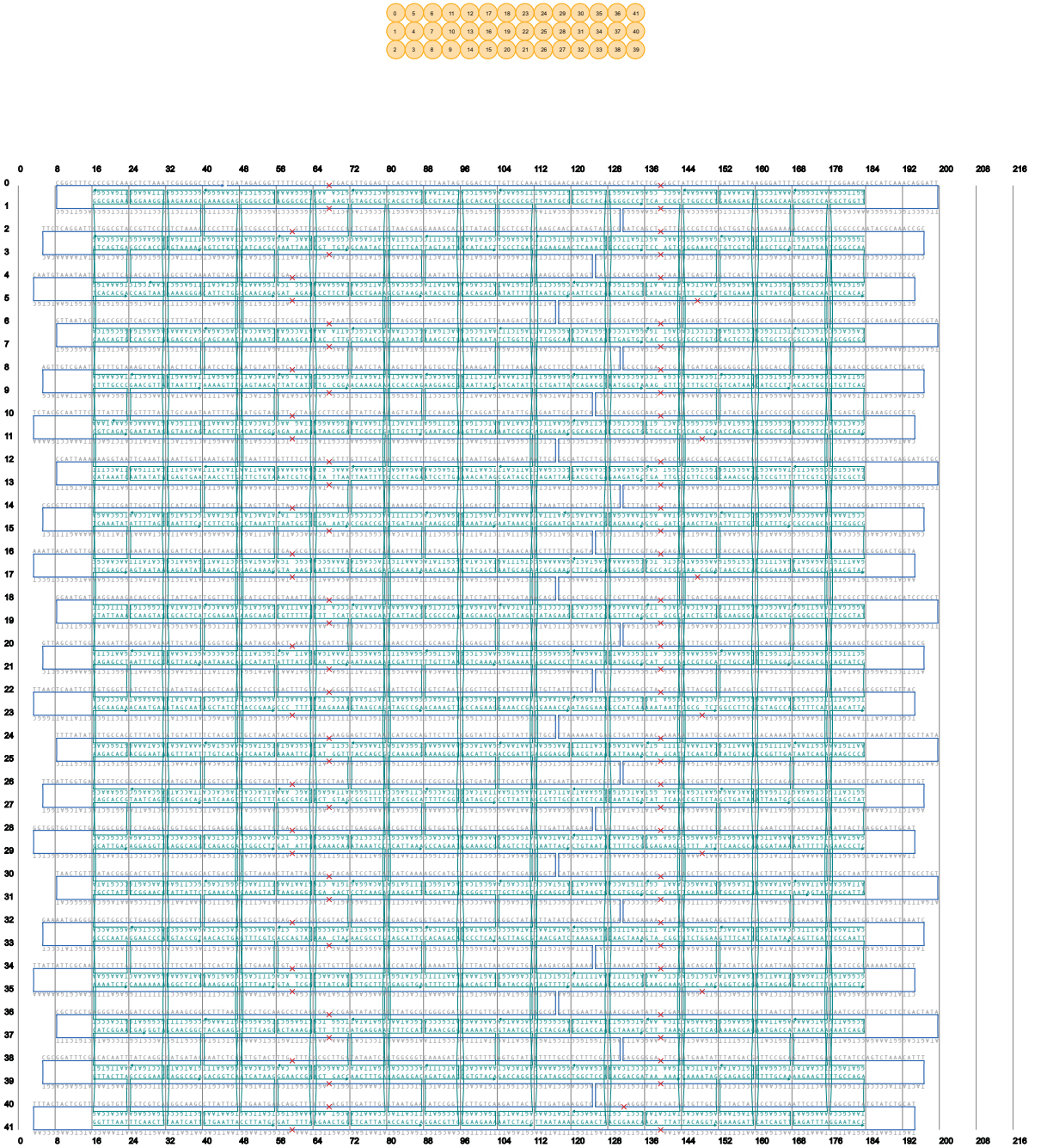


Figure S14: Design of the λ DNA Origami Structure

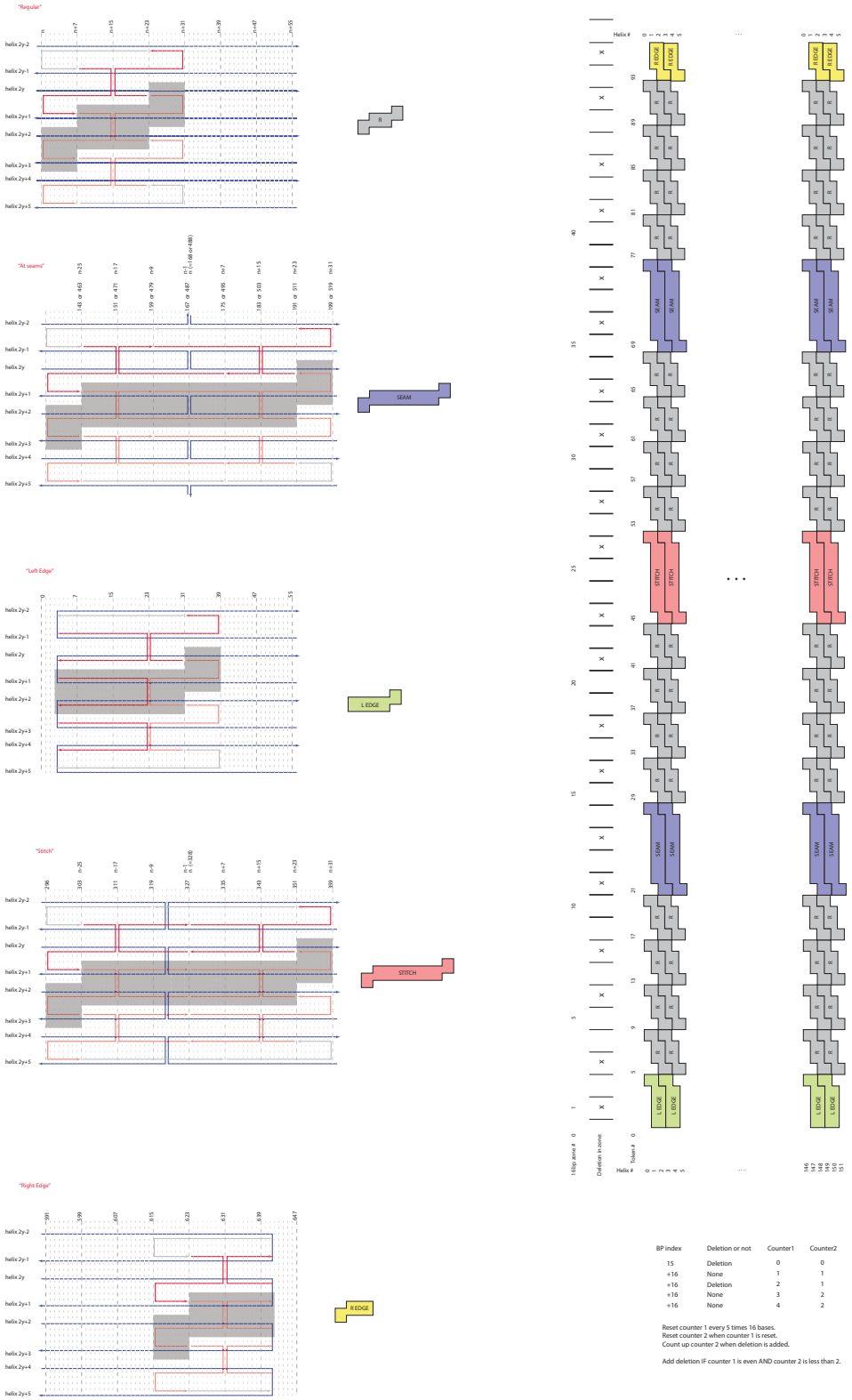


Figure S15: Truncated λ DNA Origami Design

The caDNAno file generated by the python script (Appendix S1) is to big to be opened with the caDNAno software. This truncated design with only 12 double helices instead of 154 was generated to check if the design as well as the computed sequences are correct.

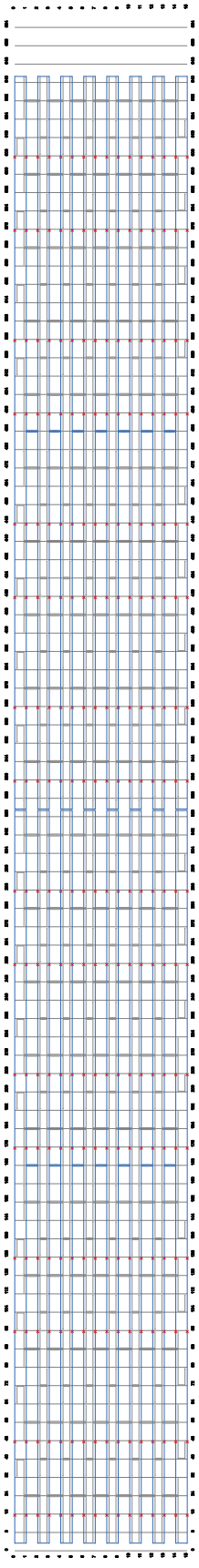
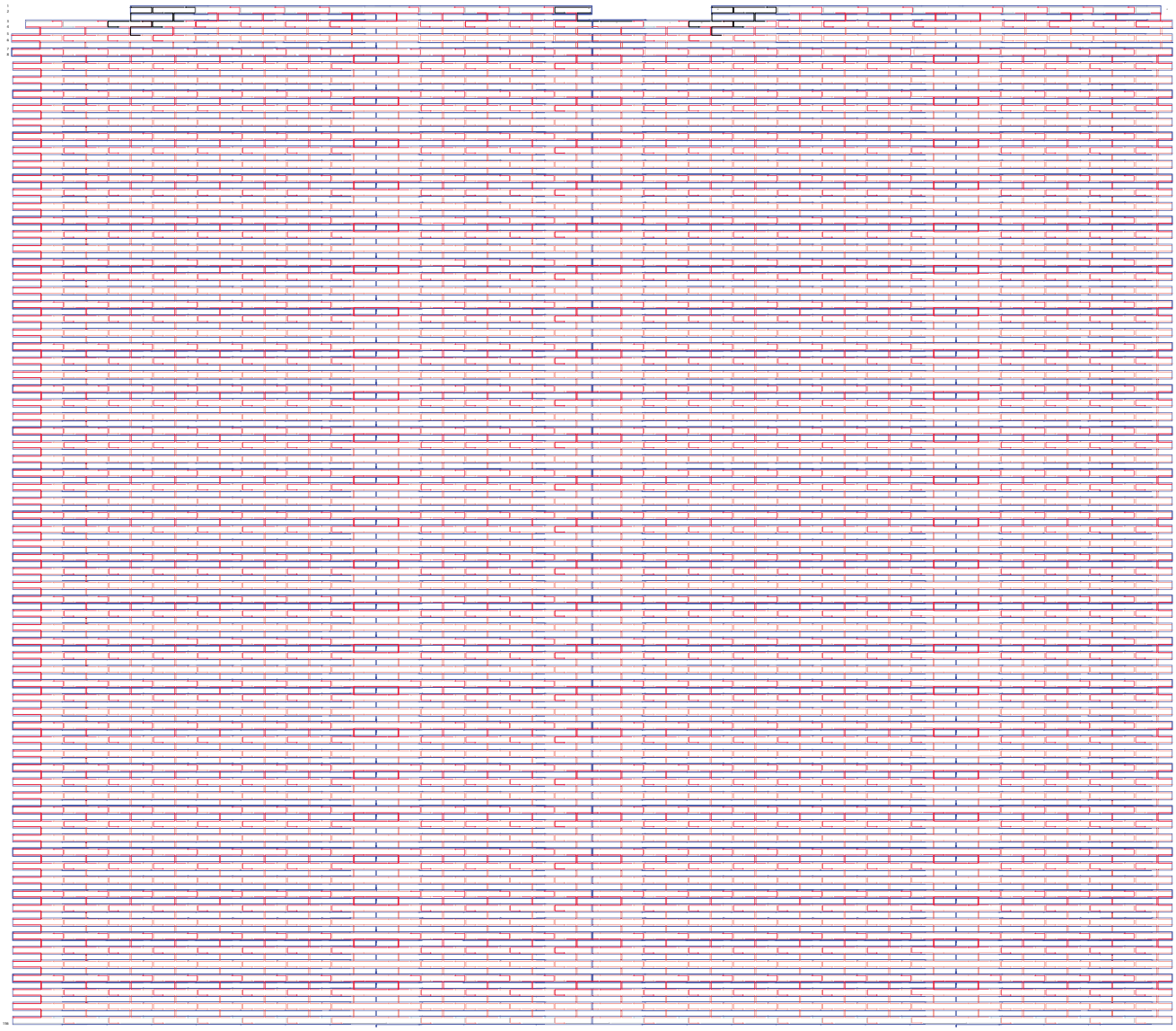


Figure S16: Entire Design Schematics of the λ DNA Origami Structure



Appendix S1: Python code used to draw the design of the λ DNA origami structure and compute the staple sequences

```
#!/usr/bin/env python
# encoding: utf-8

'''
Coded by B. Hogberg

Draws the design for the lambda double-rectangle, saves a caDNAno-compatible file and
computes the staple sequences.


For a 12 helices tall rectangle, the staples generated by caDNAno and this script matches
perfectly.
'''

# Python standard libraries
import string
import os
import sys

import caDNAnoFileHandlerClass as caDNAno

def caDNAnifyScaffold(fragments,VirtStrands):
    # Describing each fragment like this:
    # [ [at this helix,at this base], [coming from helix,base], [at this helix,go to this
base],[continued by this helix, this base] ]
    #
    # So a list of fragments will be:
    # [ [[helix1,base1], [fromHelix1,fromBase1], [goToHelix1,goToBase1], [toHelix1,
toBase1] ],
    #   [[helix2,base2], [fromHelix2,fromBase2], [goToHelix2,goToBase2], [toHelix2,
toBase2] ] ]
    #
    for frag in fragments:
        currStrandIndex=frag[0][0]
        nextStrandIndex=frag[2][0]
        prevStrandIndex=frag[1][0]
        if currStrandIndex%2==0:
            evenStrand=True
        else:
            evenStrand=False
        currBpIndex=frag[0][1]
        if evenStrand:
            nextBpIndex=frag[0][1]+1
        else:
            nextBpIndex=frag[0][1]-1
        prevBpIndex=frag[1][1]
        # Defining indecies to loop through if even strand left to right
        # if odd strand right to left.
        if evenStrand:
            indecies=range(frag[0][1],frag[2][1])
        else:
            indecies=range(frag[2][1],frag[0][1])[::-1]
        for bp in indecies:
            VirtStrands[currStrandIndex]
[currBpIndex]=[prevStrandIndex,prevBpIndex,nextStrandIndex,nextBpIndex]
            prevStrandIndex=currStrandIndex
            prevBpIndex=currBpIndex
            currBpIndex=nextBpIndex
            if evenStrand:
                nextBpIndex=currBpIndex+1
            else:
                nextBpIndex=currBpIndex-1
        # Last bp:
        VirtStrands[frag[2][0]][frag[2][1]]=[prevStrandIndex,prevBpIndex,frag[3]
[0],frag[3][1]]
```

```
def caDNAnifyStaples(fragments,VirtStrands,printOut=False):
    #
    #
    # Same as above but with changed directions of odd and even strands
    #
    #
    #
    #
    for frag in fragments:
        currStrandIndex=frag[0][0]
        nextStrandIndex=frag[2][0]
        prevStrandIndex=frag[1][0]
        if currStrandIndex%2==0:
            evenStrand=True
            if not frag[0][1]>frag[2][1]:
                print 'Strand in wrong direction!'
                return
        else:
            evenStrand=False
            if not frag[0][1]<frag[2][1]:
                print 'Strand in wrong direction!'
                return
        currBpIndex=frag[0][1]
        if evenStrand:
            nextBpIndex=frag[0][1]-1
        else:
            nextBpIndex=frag[0][1]+1
        prevBpIndex=frag[1][1]
        # Defining indecies to loop through if even strand left to right
        # if odd strand right to left.
        if evenStrand:
            indecies=range(frag[2][1],frag[0][1])[::-1]
        else:
            indecies=range(frag[0][1],frag[2][1])
        for bp in indecies:
            VirtStrands[currStrandIndex]
[currBpIndex]=[prevStrandIndex,prevBpIndex,nextStrandIndex,nextBpIndex]
            if printOut:
                print [prevStrandIndex,prevBpIndex,nextStrandIndex,nextBpIndex]
                print VirtStrands[currStrandIndex][currBpIndex]
            prevStrandIndex=currStrandIndex
            prevBpIndex=currBpIndex
            currBpIndex=nextBpIndex
            if evenStrand:
                nextBpIndex=currBpIndex-1
            else:
                nextBpIndex=currBpIndex+1
        # Last bp:
        VirtStrands[frag[2][0]][frag[2][1]]=[prevStrandIndex,prevBpIndex,frag[3]
[0],frag[3][1]]
        if printOut:
            print VirtStrands[currStrandIndex][currBpIndex]

###############
#
#   Variables
#
###############
lastBpIndex=671
lastStrandIndex=155
lftBrPt=[0,168] # 5'-end of strand is at [helix, base]
rghtBrPt=[lastStrandIndex,487]


#########################
#
```

```
# Some Initialization...
#
#########################
obj=caDNAno.caDNAnoFileHandler()
obj.initializeEmpty()
# Make empty virtual scaffold strands:
scaffVirtStrands=[[[-1,-1,-1,-1] for i in range(lastBpIndex+1)] for s in
range(lastStrandIndex+1)]
# Make empty virtual staple strands:
stapVirtStrands=[[[-1,-1,-1,-1] for i in range(lastBpIndex+1)] for s in
range(lastStrandIndex+1)]



#######################################################
#
# Scaffold Routing
#
#######################################################

# Make a list of deletion positions
bp=-1
deletions=[]
while bp+5*16 < lastBpIndex:
    insertedDeletions=0
    for i in range(5):
        bp=bp+16
        if i%2==0 and insertedDeletions<2:
            deletions.append(bp)
            insertedDeletions+=1
skipMaster=[0 for i in range(lastBpIndex+1)]
for deletion in deletions:
    skipMaster[deletion]=1
skipVirtStrands=[skipMaster for i in range(lastStrandIndex+1)]


# First two strand fragments:
firstStrandFragments0=[[  [0,88],  [1,88],   [0,322],[1,322]],
                       [[0,408],[1,408],   [0,631],[1,631]]]
caDNAnifyScaffold(firstStrandFragments0,scaffVirtStrands)
firstStrandFragments1=[[[1,631],[0,631], [1,488], [2,488]],
                       [[1,487],[2,487], [1,408], [0,408]],
                       [[1,322],[0,322], [1,168], [2,168]],
                       [[1,167],[2,167], [1,88],  [0,88]]]
caDNAnifyScaffold(firstStrandFragments1,scaffVirtStrands)

# Last strand fragments:
lastStrandFragments=[[[lastStrandIndex,642],[lastStrandIndex-1,642], [lastStrandIndex,
323],[lastStrandIndex-1,323]],
                     [[lastStrandIndex,322],[lastStrandIndex-1,322], [lastStrandIndex,3],
[lastStrandIndex-1,3]]]

caDNAnifyScaffold(lastStrandFragments,scaffVirtStrands)

# All the other strands
for s in range(2,lastStrandIndex):
    if s%2==0:
        # Even strands
        betweenFragments=[
                          [[s,3],   [s+1,3],   [s,167], [s-1,167]],
                          [[s,168], [s-1,168], [s,322], [s+1,322]],
                          [[s,323], [s+1,323], [s,487], [s-1,487]],
                          [[s,488], [s-1,488], [s,642], [s+1,642]]
                         ]
    else:   # Odd strands
```

```
        betweenFragments=[
                          [[s,642], [s-1,642], [s,488], [s+1,488]],
                          [[s,487], [s+1,487], [s,323], [s-1,323]],
                          [[s,322], [s-1,322], [s,168], [s+1,168]],
                          [[s,167], [s+1,167], [s,3],   [s-1,3]]
                         ]
    caDNAnifyScaffold(betweenFragments,scaffVirtStrands)

# Insert breakpoints
scaffVirtStrands[lftBrPt[0]][lftBrPt[1]][0]=-1
scaffVirtStrands[lftBrPt[0]][lftBrPt[1]][1]=-1
if lftBrPt[0]%2==0:
    mod=-1
else:
    mod=+1
scaffVirtStrands[lftBrPt[0]][lftBrPt[1]+mod][2]=-1
scaffVirtStrands[lftBrPt[0]][lftBrPt[1]+mod][3]=-1

scaffVirtStrands[rghtBrPt[0]][rghtBrPt[1]][0]=-1
scaffVirtStrands[rghtBrPt[0]][rghtBrPt[1]][1]=-1
if rghtBrPt[0]%2==0:
    mod=-1
else:
    mod=+1
scaffVirtStrands[rghtBrPt[0]][rghtBrPt[1]+mod][2]=-1
scaffVirtStrands[rghtBrPt[0]][rghtBrPt[1]+mod][3]=-1

#####################################################
#
# Staple strands
#
#####################################################


# Regular ones:
regularStartBases=[5*8,9*8,13*8,25*8,29*8,33*8,45*8,49*8,53*8,65*8,69*8,73*8]
for startBase in regularStartBases:
    n=startBase

    topStaple=[
                [[0,n+23],[-1,-1],[0,n],[1,n]],
                [[1,n],[0,n],[1,n+7],[-1,-1]]
                ]
    caDNAnifyStaples(topStaple,stapVirtStrands)

    bottomStaple=[
                  [[lastStrandIndex,n+8],[-1,-1],[lastStrandIndex,n+31],
[lastStrandIndex-1,n+31]],
                  [[lastStrandIndex-1,n+31],[lastStrandIndex,n+31],[lastStrandIndex-1,n
+24],[-1,-1]]
                  ]
    caDNAnifyStaples(bottomStaple,stapVirtStrands)

    for y in range(2,lastStrandIndex,2):
        n=startBase
        staple1=[
                 [[y,n+23],   [-1,-1],    [y,n+16],    [y-1,n+16]],
                 [[y-1,n+16], [y,n+16],   [y-1,n+31],  [y-2,n+31]],
                 [[y-2,n+31], [y-1,n+31], [y-2,n+24],  [-1,-1]]
                 ]
        caDNAnifyStaples(staple1,stapVirtStrands)

        staple2=[
                 [[y-1,n+8],[-1,-1],    [y-1,n+15],   [y,n+15]],
                 [[y,n+15], [y-1,n+15], [y,n],        [y+1,n]],
                 [[y+1,n],  [y,n],      [y+1,n+7],    [-1,-1]]
```

```
                ]
        caDNAnifyStaples(staple2,stapVirtStrands)

# Left edge
topLeftCorner=[[[2,31],[-1,-1],[2,3],[-1,-1]]]
caDNAnifyStaples(topLeftCorner,stapVirtStrands)

botLeftCorner=[
               [[lastStrandIndex,3],[-1,-1],[lastStrandIndex,39],[lastStrandIndex-1,39]],
               [[lastStrandIndex-1,39],[lastStrandIndex,39],[lastStrandIndex-1,32],
[-1,-1]]
               ]
caDNAnifyStaples(botLeftCorner,stapVirtStrands)

for y in range(4,lastStrandIndex,2):
    n=8
    snakeStaple=[
             [[y,n+23],   [-1,-1],    [y,n+16],    [y-1,n+16]],
             [[y-1,n+16], [y,n+16],   [y-1,n+31],  [y-2,n+31]],
             [[y-2,n+31], [y-1,n+31], [y-2,n+24],  [-1,-1]]
             ]
    caDNAnifyStaples(snakeStaple,stapVirtStrands)

    stapleStaple=[
                  [[y-1,3],  [-1,-1],  [y-1,23], [y,23]],
                  [[y,23],   [y-1,23], [y,3],  [-1,-1]]
                  ]
    caDNAnifyStaples(stapleStaple,stapVirtStrands)

# At seams
seamPositions=[168,488]
for n in seamPositions:

    top1 = [ [[0,n+23],[-1,-1],[0,n-8],[-1,-1]] ]
    caDNAnifyStaples(top1,stapVirtStrands)

    top2 = [
            [[0,n-9],[-1,-1],[0,n-32],[1,n-32]],
            [[1,n-32],[0,n-32],[1,n-25],[-1,-1]]
           ]
    caDNAnifyStaples(top2,stapVirtStrands)

    bot1 = [ [[lastStrandIndex,n-24],[-1,-1],[lastStrandIndex,n+7],[-1,-1]] ]
    caDNAnifyStaples(bot1,stapVirtStrands)

    bot2 = [
             [[lastStrandIndex,n+8],[-1,-1],[lastStrandIndex,n+31],[lastStrandIndex-1,n
+31]],
             [[lastStrandIndex-1,n+31],[lastStrandIndex,n+31],[lastStrandIndex-1,n+24],
[-1,-1]]
           ]
    caDNAnifyStaples(bot2,stapVirtStrands)

    for y in range(2,lastStrandIndex,2):
        leftSnake=[
                  [[y-1,n-24], [-1,-1],      [y-1,n-17],   [y,n-17]],
                  [[y,n-17],   [y-1,n-17],   [y,n-32],     [y+1,n-32]],
                  [[y+1,n-32], [y,n-32],     [y+1,n-25],   [-1,-1]]
                 ]
        caDNAnifyStaples(leftSnake,stapVirtStrands)

        leftRound=[
                    [[y, n+7],    [-1, -1],   [y, n-16],   [y-1, n-16]],
                    [[y-1, n-16], [y, n-16],  [y-1, n-9],  [-1, -1]]
                   ]
        caDNAnifyStaples(leftRound,stapVirtStrands)
```

```
        rightRound=[
                    [[y-1,n-8],[-1,-1],[y-1,n+15],[y,n+15]],
                    [[y,n+15],[y-1,n+15],[y,n+8],[-1,-1]]
                    ]
        caDNAnifyStaples(rightRound,stapVirtStrands)

        rightSnake=[
                [[y,n+23],   [-1,-1],    [y,n+16],    [y-1,n+16]],
                [[y-1,n+16], [y,n+16],   [y-1,n+31],  [y-2,n+31]],
                [[y-2,n+31], [y-1,n+31], [y-2,n+24],  [-1,-1]]
                ]
        caDNAnifyStaples(rightSnake,stapVirtStrands)

# Stitch
n=328

top1 = [ [[0,n+23],[-1,-1],[0,n-8],[-1,-1]] ]
caDNAnifyStaples(top1,stapVirtStrands)

top2 = [
        [[0,n-9],[-1,-1],[0,n-32],[1,n-32]],
        [[1,n-32],[0,n-32],[1,n-25],[-1,-1]]
       ]
caDNAnifyStaples(top2,stapVirtStrands)

bot1 = [ [[lastStrandIndex,n-24],[-1,-1],[lastStrandIndex,n+7],[-1,-1]] ]
caDNAnifyStaples(bot1,stapVirtStrands)

bot2 = [
        [[lastStrandIndex,n+8],[-1,-1],[lastStrandIndex,n+31],[lastStrandIndex-1,n+31]],
        [[lastStrandIndex-1,n+31],[lastStrandIndex,n+31],[lastStrandIndex-1,n+24],
[-1,-1]]
       ]
caDNAnifyStaples(bot2,stapVirtStrands)
for y in range(2,lastStrandIndex,2):
    leftSnake=[
              [[y-1,n-24], [-1,-1],      [y-1,n-17],   [y,n-17]],
              [[y,n-17],   [y-1,n-17],   [y,n-32],     [y+1,n-32]],
              [[y+1,n-32], [y,n-32],     [y+1,n-25],   [-1,-1]]
             ]
    caDNAnifyStaples(leftSnake,stapVirtStrands)

    leftRound=[
               [[y, n-1],    [-1, -1],   [y, n-16],   [y-1, n-16]],
               [[y-1, n-16], [y, n-16],  [y-1, n-1],  [-1, -1]]
              ]
    caDNAnifyStaples(leftRound,stapVirtStrands)

    rightRound=[
                [[y-1,n],[-1,-1],[y-1,n+15],[y,n+15]],
                [[y,n+15],[y-1,n+15],[y,n],[-1,-1]]
                ]
    caDNAnifyStaples(rightRound,stapVirtStrands)

    rightSnake=[
             [[y,n+23],   [-1,-1],    [y,n+16],    [y-1,n+16]],
             [[y-1,n+16], [y,n+16],   [y-1,n+31],  [y-2,n+31]],
             [[y-2,n+31], [y-1,n+31], [y-2,n+24],  [-1,-1]]
             ]
    caDNAnifyStaples(rightSnake,stapVirtStrands)

# Right edge

topRightCorner=[
                [[0,642],[-1,-1],[0,616],[1,616]],
```

```
                [[1,616],[0,616],[1,623],[-1,-1]]
                ]
caDNAnifyStaples(topRightCorner,stapVirtStrands)

botRightCorner=[ [[lastStrandIndex,624],[-1,-1],[lastStrandIndex,642],[-1,-1]] ]
caDNAnifyStaples(botRightCorner,stapVirtStrands)


for y in range(2,lastStrandIndex,2):
    n=616
    snake=[
             [[y-1,n+8],[-1,-1],    [y-1,n+15],   [y,n+15]],
             [[y,n+15], [y-1,n+15], [y,n],        [y+1,n]],
             [[y+1,n],  [y,n],      [y+1,n+7],    [-1,-1]]
            ]
    caDNAnifyStaples(snake,stapVirtStrands)

    stapleStaple=[
                  [[y,642],[-1,-1],[y,632],[y-1,632]],
                  [[y-1,632],[y,632],[y-1,642],[-1,-1]]
                    ]
    caDNAnifyStaples(stapleStaple,stapVirtStrands)

# Erase staples at top to make room for notches.
# Then edit the bases that needs change.
for i in range(88):
    stapVirtStrands[0][i]=[-1,-1,-1,-1]
    stapVirtStrands[1][i]=[-1,-1,-1,-1]
for i in range(323,408):
    stapVirtStrands[0][i]=[-1,-1,-1,-1]
    stapVirtStrands[1][i]=[-1,-1,-1,-1]

# A
notchChangeA=[ [[2,63],[-1,-1],[2,55],[2,54]] ]
caDNAnifyStaples(notchChangeA,stapVirtStrands)
# B
stapVirtStrands[2][87]=[-1,-1,2,86]
# C
notchChangeB=[ [[0,103],[1,103],[0,88],[-1,-1]] ]
caDNAnifyStaples(notchChangeB,stapVirtStrands)
# D
notchChangeD=[ [[0,322],[-1,-1],[0,296],[1,296]] ]
caDNAnifyStaples(notchChangeD,stapVirtStrands)
# E
notchChangeE=[ [[1,312],[2,312],[1,322],[-1,-1]] ]
caDNAnifyStaples(notchChangeE,stapVirtStrands)
# F
notchChangeF=[ [[2,351],[-1,-1],[2,328],[-1,-1]] ]
caDNAnifyStaples(notchChangeF,stapVirtStrands)
# G
notchChangeG=[ [[2,407],[-1,-1],[2,392],[3,392]] ]
caDNAnifyStaples(notchChangeG,stapVirtStrands)
# H
notchChangeH=[ [[0,423],[1,423],[0,408],[-1,-1]] ]
caDNAnifyStaples(notchChangeH,stapVirtStrands)
# I
notchChangeI=[ [[2,383],[-1,-1],[2,360],[3,360]] ]
caDNAnifyStaples(notchChangeI,stapVirtStrands)

# J - scaffold
for i in range(13):
    scaffVirtStrands[2][i]=[-1,-1,-1,-1]
    scaffVirtStrands[3][i]=[-1,-1,-1,-1]
scaffVirtStrands[2][13]=[3,13,2,14]
scaffVirtStrands[3][13]=[3,12,2,13]
# J - staples
```

```
for i in range(13):
    stapVirtStrands[2][i]=[-1,-1,-1,-1]
    stapVirtStrands[3][i]=[-1,-1,-1,-1]
stapVirtStrands[2][13]=[2,12,-1,-1]
stapVirtStrands[3][13]=[-1,-1,3,14]

# K - scaffold
for i in range(632,643):
    scaffVirtStrands[0][i]=[-1,-1,-1,-1]
    scaffVirtStrands[1][i]=[-1,-1,-1,-1]
scaffVirtStrands[0][631]=[0,630,1,631]
scaffVirtStrands[1][631]=[0,631,1,630]
# K - staples
for i in range(632,643):
    stapVirtStrands[0][i]=[-1,-1,-1,-1]
    stapVirtStrands[1][i]=[-1,-1,-1,-1]
stapVirtStrands[0][631]=[-1,-1,0,630]
stapVirtStrands[2][632]=[2,633,-1,-1]


##################################################
#
# Writing caDNAno compatible file
#
##################################################
for i in range(lastStrandIndex+1):

obj.appendStrand(num=i,scaf=scaffVirtStrands[i],stap=stapVirtStrands[i],skip=skipVirtStra
nds[i],row=i,col=1)

#obj.printBasicData()
obj.writeCaDNAnoFile('lambdaRectangle.json')



##################################################
#
#  Printing out the staple sequences
#
##################################################
def reverse(sequence):
    """Returns the bases in reverse order"""
    return sequence[::-1]

def complement_to(sequence):
    """Returns the wc-complement to the sequence"""
    comp = {'a':'T', 'A':'T', 'c':'G', 'C':'G', 'g':'C', 'G':'C', 't':'A', 'T':'A',
'D':'D', '?':'?'}
    temp_complement = ''
    for letter in sequence:
        if letter in comp:
            temp_complement = temp_complement + comp[letter]
    return reverse(temp_complement)

def stripped_seq(raw_sequence):
    """Clean the input sequence from unnecessary characters
       and makes the bases uppercase"""
    uppercase = {'a':'A', 'A':'A', 'c':'C', 'C':'C', 'g':'G', 'G':'G', 't':'T', 'T':'T'}
    temp_seq_ra = ''
    for letter in raw_sequence:
        if letter in uppercase:
            temp_seq_ra = temp_seq_ra + uppercase[letter]
    return temp_seq_ra

# First populate the virtual scaffol strands:
```

```
# make an empty ('?') set of virtual strands
wSequVirtStrs=[['?' for i in range(lastBpIndex+1)] for s in range(lastStrandIndex+1)]

input_file = file('LambdaForwardStrand.txt', 'r')
scaffoldLeftStr  = stripped_seq(input_file.read())
input_file.close()
input_file2 = file('LambdaReverseStrand.txt','r')
scaffoldRightStr = stripped_seq(input_file2.read())
input_file2.close()

scaffoldLeft=list(scaffoldLeftStr)
scaffoldRight=list(scaffoldRightStr)

# Left Part:
lenBefore=len(scaffoldLeft)
currBase=scaffVirtStrands[lftBrPt[0]][lftBrPt[1]]
currBaseIn=lftBrPt[1]
currStrIn=lftBrPt[0]
lastPointAdded=False
while not lastPointAdded:
    if skipVirtStrands[currStrIn][currBaseIn]==1:
        # Here a deletion should be added, marked by 'D'
        wSequVirtStrs[currStrIn][currBaseIn]='D'
    else:
        wSequVirtStrs[currStrIn][currBaseIn]=scaffoldLeft.pop(0)
    currBaseIn=currBase[3]
    currStrIn=currBase[2]
    if currBaseIn==-1 and currStrIn==-1:
        lastPointAdded=True
    else:
        currBase=scaffVirtStrands[currStrIn][currBaseIn]
lenAfter=len(scaffoldLeft)
print 'In left part '+str(lenBefore-lenAfter)+' bases were added and '+str(lenAfter)+'
bases are left over'

# Right Part:
lenBefore=len(scaffoldRight)
currBase=scaffVirtStrands[rghtBrPt[0]][rghtBrPt[1]]
currBaseIn=rghtBrPt[1]
currStrIn=rghtBrPt[0]
lastPointAdded=False
while not lastPointAdded:
    if skipVirtStrands[currStrIn][currBaseIn]==1:
        # Here a deletion should be added, marked by 'D'
        wSequVirtStrs[currStrIn][currBaseIn]='D'
    else:
        wSequVirtStrs[currStrIn][currBaseIn]=scaffoldRight.pop(0)
    currBaseIn=currBase[3]
    currStrIn=currBase[2]
    if currBaseIn==-1 and currStrIn==-1:
        lastPointAdded=True
    else:
        currBase=scaffVirtStrands[currStrIn][currBaseIn]
lenAfter=len(scaffoldRight)
print 'In right part '+str(lenBefore-lenAfter)+' bases were added and '+str(lenAfter)+'
bases are left over'



#####################################################################
#       OUTPUT STAPLE STRANDS
# Go through the virtual staple strands and look for
# staple startpoints. Horizontally, through each of the two
# sub-structures.
#####################################################################
stapBcount=0
```

```
stapCount=0
output=[]
strippedOut=[]
rows=['A','B','C','D','E','F','G','H']
plateNum=774
for subStructure in range(2):
    if subStructure==0:
        print 'Forward strand staples starting w. index '+str(stapCount)
        output.append(',,Forward Strand\n')
        strtBase=0
        endBase=323
    else:
        print 'Reverse strand staples starting w. index '+str(stapCount)
        output.append(',,Reverse Strand\n')
        strtBase=328
        endBase=643
    plateNum+=1
    r=-1
    for sI in range(lastStrandIndex+1):
        r+=1
        if r==8: # Reset row numbering and start new plate
            plateNum+=1
            r=0
        c=1  # Reset column numbering
        for bI in range(strtBase,endBase):
            if stapVirtStrands[sI][bI][0]==-1 and stapVirtStrands[sI][bI][1]==-1 and
stapVirtStrands[sI][bI][2]>-1:
                # A staple start point
                stapleSeq=[]
                currBase=stapVirtStrands[sI][bI]
                currBaseIn=bI
                currStrIn=sI
                lastPointAdded=False
                while not lastPointAdded:
                    if skipVirtStrands[currStrIn][currBaseIn]==0: # only add bases if no
deletion
                        stapleSeq.append( complement_to(wSequVirtStrs[currStrIn]
[currBaseIn]) )
                        stapBcount+=1
                    currBaseIn=currBase[3]
                    currStrIn=currBase[2]
                    if currBaseIn==-1 and currStrIn==-1:
                        lastPointAdded=True
                    else:
                        currBase=stapVirtStrands[currStrIn][currBaseIn]
                stringoligo=''.join(stapleSeq)
                output.append(str(plateNum)+','+rows[r]+str(c)+','+stringoligo+'\n')
                strippedOut.append(stringoligo+'\n')
                stapCount+=1
                c+=1
                if c==14:
                    print 'Warning, col number was too high in plate '+str(plateNum)
        while c<13:
            # fill all columns
            output.append(str(plateNum)+','+rows[r]+str(c)+','+'\n')
            c+=1

# Connector staples
print 'Connector staples starting w. index '+str(stapCount)
plateNum+=1
r=0
c=0
output.append(',,Connector staples\n')
for sI in range(lastStrandIndex+1):
    if stapVirtStrands[sI][327][0]==-1 and stapVirtStrands[sI][327][1]==-1 and
stapVirtStrands[sI][327][2]>-1:
```

```
        # A staple start point
        c+=1
        if c==13:
            c=1
            r+=1
        stapleSeq=[]
        currBase=stapVirtStrands[sI][327]
        currBaseIn=327
        currStrIn=sI
        lastPointAdded=False
        while not lastPointAdded:
            if skipVirtStrands[currStrIn][currBaseIn]==0: # only add bases if no deletion
                stapleSeq.append( complement_to(wSequVirtStrs[currStrIn][currBaseIn]) )
                stapBcount+=1
            currBaseIn=currBase[3]
            currStrIn=currBase[2]
            if currBaseIn==-1 and currStrIn==-1:
                lastPointAdded=True
            else:
                currBase=stapVirtStrands[currStrIn][currBaseIn]
        stringoligo=''.join(stapleSeq)
        output.append(str(plateNum)+','+rows[r]+str(c)+','+stringoligo+'\n')
        strippedOut.append(stringoligo+'\n')
        stapCount+=1

print 'The total number of staple bases calculated is '+str(stapBcount)+' on a total of
'+str(stapCount)+' staple strands.'


output_file=file('./LambdaStaples.cvs','w')
output_file.write(''.join(output))
output_file.close()

output_file2=file('./strippedLambdaStaples.txt','w')
output_file2.write(''.join(strippedOut))
output_file2.close()
```