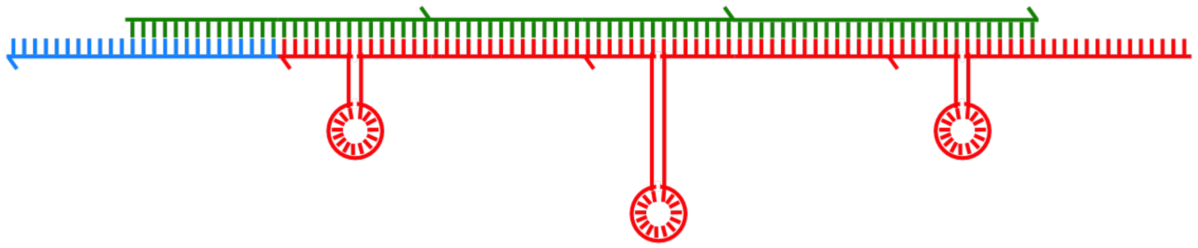
dnastack

Release 0.1

Ben Shirt-Ediss

Aug 11, 2021

CONTENTS:



This documentation describes how to install and run **dnastack**, a stochastic model of the the DNA stack chemistry described in paper: A Last-In First-Out Stack Data Structure Implemented in DNA, Lopiccolo and Shirt-Ediss, et al., Nature Comms 12, 4861 (2021) doi:10.1038/s41467-021-25023-6.

dnastack is written in Python 3 and makes use of the stocal simulation engine for polymer chemistries that have a variable size state space. The simulation outputs the predicted concentration of each DNA polymer species at the end of an experiment and these concentrations can be further displayed as virtual polyacrylamide gel images.

The source code for the model is available on Bitbucket, released under an MIT Licence.

CHAPTER

ONE

INSTALLATION

Typical installation time is 10 minutes.

1.1 Mac OS

Installation is via the terminal. Open Terminal.app (in Applications/Utilities).

1. Change to the directory where you want to put the source code of **dnastack**. For example:

```
cd /Users/myname
```

2. Download a copy of the source code, using the following command:

```
git clone https://bitbucket.org/engineering-data-structure-organoids/dnastack.git
```

Note: Running the `git` command for the first time on Mac OS may require you to download developer tools. If these are not installed, a dialog box should prompt you to install them.

3. Change to the newly created `dnastack` directory:

```
cd dnastack
```

4. Make a new python3 virtual environment, enable it and install the required packages:

```
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

5. Run the program by following the *Quick Demo* page, or the more detailed instructions on the *Running a Simulation* page.

6. Done? Leave the virtual environment by typing:

```
deactivate
```

If you would like to make virtual gel images of simulation output (see page *Virtual Gel Image*), follow the additional steps below to install ImageMagick.

7. Install the Mac OS package manager Homebrew. This is necessary for installing ImageMagick in the next step. As described at https://brew.sh, install Homebrew by running command:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install.sh)"
```

8. Install ImageMagick using homebrew:

```
brew install imagemagick
```

1.2 Linux Ubuntu

Installation is via the terminal.

1. Update your repositories and install git:

```
sudo apt-get update
sudo apt install git
```

2. Install pip for Python3:

```
sudo apt-get install build-essential libssl-dev libffi-dev python-dev
sudo apt install python3-pip
```

3. Use pip to install virtualenv:

```
sudo pip3 install virtualenv
```

4. If you would like to make virtual gel images of simulation output (see page *Virtual Gel Image*), install ImageMagick:

```
sudo apt install imagemagick
```

5. Follow the same instructions as for Mac OS above, up to Mac OS step 6.

1.3 Windows 10

We recommend installing a Linux Ubuntu environment on Windows, and then following the Linux installation instructions above. A Linux environment can be installed by using the Windows Subsystem for Linux (available on 64-bit versions of Windows 10). Alternatively, using Ubuntu under VirtualBox is another option.

CHAPTER

TWO

QUICK DEMO

To run a quick demo to test that the installation works:

1. Change to the dnastack directory and activate the Python virtual environment (if not already activated):

```
cd dnastack
source venv/bin/activate
```

2. Then run a stochastic simulation of adding DNA strands in order *start* - *push* - *X* (each at 300nM concentration and separated by a wait time of 30 minutes) by typing:

```
python3 washing.py example 3
```

After around 1 minute of computation, output similar to the following should be produced:

```
------------------------------------------------------
STOCAL stochastic simulation of stack washing
spX
example lane 3

PHI_0 = 33% volume fraction of supernatant species SURVIVE each wash via non-specific␣
↪bead binding initially, when normalised bead mass = 1.0
MU  = 10% of remaining beads LOST on each wash
VOL = 1.50000e-13 litre (100nM = 9033 particles, 300nM = 27099 particles)

        Bead mass remaining: 1.00
        Incubating linker (200 nM) with beads, waiting sufficient time to get 100%␣
↪binding
        --- Wash ---
        Bead mass remaining: 0.90
        Adding s (300 nM) to reaction, waiting sufficient time to get 100% reaction␣
↪completion
        --- Wash ---
        Bead mass remaining: 0.81
        Adding p (300 nM) to reaction, waiting 30.00 mins
        (Recording chemistry state, supernatant state, and supernatant state after a␣
↪releaser)
        --- Wash ---
        Bead mass remaining: 0.73
        Adding X (300 nM) to reaction, waiting 30.00 mins
        (Recording chemistry state, supernatant state, and supernatant state after a␣
↪releaser)
```

(continues on next page)

(continued from previous page)

```

Done.


------------------------------------------------------
Final state: (not directly observable by electrophoresis)
8 nM                ks                        725
0 nM                s                        30
0 nM                p                        4
6 nM                ksp                        576
0 nM                sp                        33
123 nM                X                        11126
119 nM                kspX                        10772
0 nM                Xp                        52
0 nM                pX                        14
6 nM                spX                        569
10 nM                XpX                        982
0 nM                kspXp                        56
10 nM                kspXpX                        951
0 nM                spXp                        2
0 nM                spXpX                        58
0 nM                pXpX                        1
0 nM                XpXp                        2
0 nM                kspXpXpX                        75
0 nM                XpXpX                        72
0 nM                kspXpXp                        8
0 nM                spXpXpX                        4
0 nM                XpXpXpXpX                        1
0 nM                XpXpXpX                        5
0 nM                kspXpXpXpX                        5
0 nM                kspXpXpXpXpX                          1

Supernatant of final state:
0 nM                s                        30
0 nM                p                        4
0 nM                sp                        33
123 nM                X                        11126
0 nM                Xp                        52
0 nM                pX                        14
6 nM                spX                        569
10 nM                XpX                        982
0 nM                spXp                        2
0 nM                spXpX                        58
0 nM                pXpX                        1
0 nM                XpXp                        2
0 nM                XpXpX                        72
0 nM                spXpXpX                        4
0 nM                XpXpXpXpX                        1
0 nM                XpXpXpX                        5

Supernatant state AFTER further wash, and releaser applied: (released stacks present)
7 nM                s                        658
```

(continues on next page)

(continued from previous page)

```
3 nM                sp                        320
110 nM                spX                        10001
0 nM                spXp                        26
10 nM                spXpX                        910
0 nM                spXpXpX                        70
0 nM                spXpXp                        6
0 nM                spXpXpXpX                        4
24 nM                X                        2194
0 nM                Xp                        6
2 nM                XpX                        195
0 nM                XpXpX                        16
0 nM                XpXpXpX                        1
68 nM                z                        6219
```

The numerical data reported are the final concentrations (and particle numbers) of all species in the stack chemistry. Note that the output you obtain will probably vary slightly due to the simulation being stochastic.

See the *Running a Simulation* page for more detailed running instructions.

Also, see the *Virtual Gel Image* page for how to turn numerical simulation results into a more intuitive virtual polyacrylamide gel image.

CHAPTER

THREE

RUNNING A SIMULATION

This page contains more detailed instructions about running a DNA stack simulation.

Before running any simulation, change to the `dnastack` directory and activate the Python virtual environment:

```
cd dnastack
source venv/bin/activate
```

3.1 General Syntax

To run a stochastic simulation of gel "example", lane 2, the command is:

```
python3 washing.py example 2
```

The name of the gel is specified first, followed by the lane on the gel to simulate.

The output of the simulator is a pickle file placed in the `results` directory at the end of the simulation (in the above case, `sim_example_2.pkl` will be created). This pickle file contains the model state after *each* brick is added (see *Output Pickle File* for details).

When simulations of one or more lanes of a gel have been executed, a virtual image of the gel can be produced as described in *Virtual Gel Image*.

Note: Stochastic simulation proceeds via the Gillespie Direct algorithm, which explicitly simulates every reaction. Simulations where many strands are added one after the other can take significant time to execute (hours, or sometimes days). The system volume in `constants.py` is set to 1.5e-13 litres (0.15 picolitres) to keep the species populations at acceptable levels for computationally tractable simulations.

3.2 Defining a Gel

To define a gel to simulate, a simple python function in the file `constants.py` must be created that has the name of the gel.

The function `example()` in `constants.py`, for instance, specifies a demo gel called "example". The code inside the function describes what DNA strands are put in each of the lanes of the gel. Looking at the code for lane 2, we see:

```
def example(lane) :
  if(lane == 2) :
```

(continues on next page)

(continued from previous page)

```
    bricks_add_order    =   ['s',    'p']
    bricks_add_conc     =   [300e-9, 300e-9]
    bricks_wait_time    =   [    30*60,   30*60]
```

This code means that in lane 2 of "example" gel:

- a **start** strand is initially added at 300nM…
- then 30*60 seconds are waited (30 minutes)…
- after which a **push** strand is added at 300nM..
- followed by a wait of a further 30 minutes.

3.3 Specifying Washing Parameters

If not specified, the simulation uses the default `MU` and `PHI` washing parameters, specified in `constants.py`.

- `PHI` is the initial fraction of supernatant species transferred through the wash to the next reaction (between 0 and 1, inclusive).
- `MU` is the fraction of beads lost on each wash (between 0 and 1, inclusive)

For convenience, these two parameters can also be specified on the command line, overriding the default values in `constants.py`. For example, to specify `PHI` = 0.2, `MU` = 0.1 use:

```
python3 washing.py example 2 0.2 0.1
```

CHAPTER

FOUR

OUTPUT PICKLE FILE

On completion, the simulator outputs a pickle file of results for each gel lane (i.e. each individual experiment) to the `results` directory. Pickle is a format used by Python to save data. This page explains the format of the pickle data file.

You can load a pickle file by writing the following Python script:

```
import pickle
stackdata = pickle.load( open( "my_results_file.pkl", "rb" ) )
```

Each pickle file contains a dictionary with the following keys. Underneath each key is described the attribute that it maps to.

4.1 Dictionary keys to general properties of the simulation

stackdata["bricks_add_order"]

List detailing the order in which strands were added to solution. E.g.

```
['s', 'p', 'X']
```

stackdata["bricks_add_conc"]

List of Molar concentrations that strands above were added at. E.g.

```
[3e-07, 3e-07, 3e-07]
```

for 300nM concentrations.

stackdata["bricks_wait_time"]

List of reaction wait times (seconds) that elapsed *after* each strand above was added, before the next washing step was carried out. E.g.

```
[1800, 1800, 1800]
```

for 30 minute wait times.

stackdata["rule_name"]

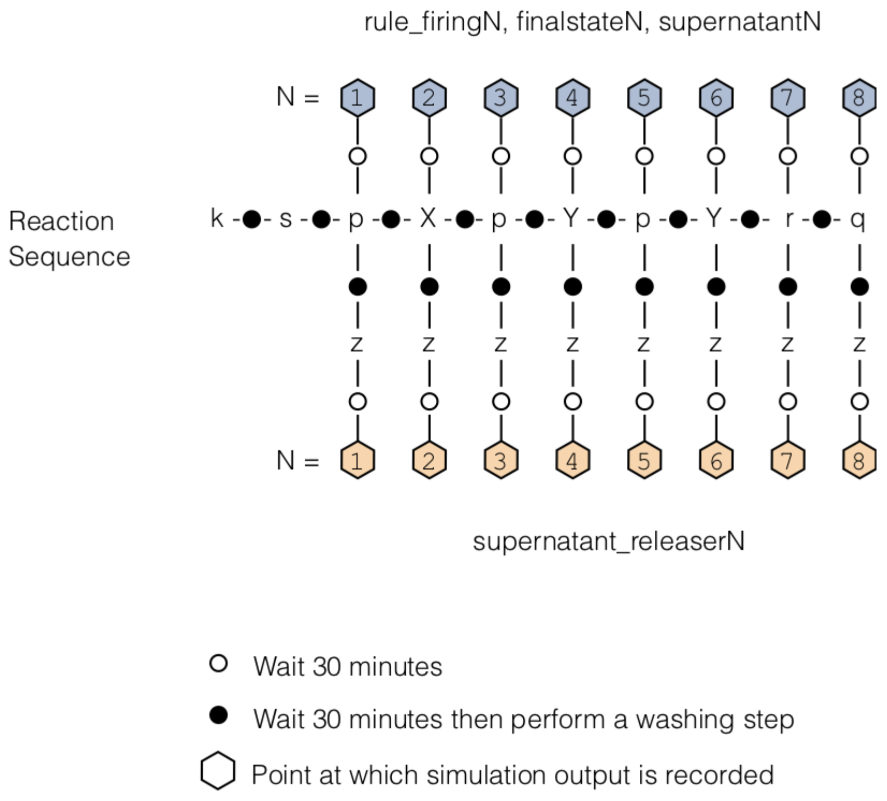
List of reaction rule names in the reaction model describing the DNA stack chemistry.

```
['LH1', 'LH2', 'LH3', 'H1', 'H2', 'LH4', 'LH5', 'LH6', 'SD1', 'SD2', 'SD3', 'SD4', 'SD5',
↪ 'SD6', 'SD7', 'SD8', 'C1', 'C2', 'C3', 'C4']
```

4.2 Dictionary keys to chemistry state at each simulation stage

The remaining keys in the dictionary track the state of the model chemistry as each strand is added, one after the other. The keys have a number on the end, N, which indicates that the key refers to the chemistry state when N strands have been added after the original **start** strand.

The diagram below describes which chemistry state is referred to by a key followed by a number N:


rule_firingN, finalstateN, supernatantN
N =
1
2
3
4
5
6
7
8
Reaction Sequence
k
s
p
X
p
Y
p
Y
r
q
z
N =
1
2
3
4
5
6
7
8
supernatant_releaserN
Wait 30 minutes
Wait 30 minutes then perform a washing step
Point at which simulation output is recorded

For example, keys `rule_firing1`, `finalstate1`, `supernatant1`, refer to the chemistry state after the first strand following **start** is added and a wait time elapses. Key `supernatant_releaser1`, on the other hand, refers to the chemistry state when the first strand following **start** is added and (i) a wait time elapses, (ii) a wash happens, (iii) **releaser** is added and (iv) a further wait time elapses.

Contents of the keys are described below.

stackdata["rule_firingN"]

List of the total number of times each rule has fired, after the Nth strand following **start** is added and a waiting time elapses. E.g.

```
[234, 23, 1, 0, 12, 0, 34, 0, 345, 550, 0, 1, 0, 6, 0, 0, 12, 35, 0, 0]
```

stackdata["finalstateN"]

Dictionary of the whole system state after the Nth strand following **start** has been added and a waiting time elapses. This is the state of every molecular species in the system, including all species tethered to beads and all species in supernatant.

The dictionary keys are species (strings) and the dictionary values are tuples giving (particle number, nM concentration). For example:

```
{'ks': (878, 9.719688673190916), 's': (187, 2.07013870374339 56), 'p': (10634, 117.
↪7211496021779), 'ksp': (13755, 152.27143245984172), 'sp': (2710, 30.000405813607497)}
```

stackdata["supernatantN"]

Dictionary of species existing in the *supernatant solution only*, after the Nth strand following **start** has been added and a waiting time elapses. Format same as above. E.g.

```
{'s': (187, 2.07013870374339 56), 'p': (10634, 117.7211496021779), 'sp': (2710, 30.
↪000405813607497)}
```

The supernatant does not contain any stacks tethered to beads (species starting "k").

stackdata["supernatant_releaserN"]

Dictionary of species existing in the supernatant when, after the Nth strand following **start** has been added, the system is washed, releaser is added and then a waiting time elapses.

For example:

```
{'s': (501, 5.546200484360648), 'sp': (13363, 147.93189036429408), 'p': (2225, 24.
↪63132949641206), 'z': (4897, 54.21106541300218)}
```

These species contain the stacks released from the beads, a fraction of any other supernatant species and any surplus **releaser** species ("z").

The biotinylated DNA **linker** strands that are attached to sepharose beads ("k") remain on the beads, possibly hybridised to **releaser**, and do not make it into the supernatant in this case.

Note: Kinetic trajectory information is **NOT** recorded in the results pickle file, as this tends to have an extremely large size. Only **end states** of each reaction stage are recorded.

CHAPTER

FIVE

VIRTUAL GEL IMAGE

Virtual gel images can be created for DNA stack simulations that use only **X** signals. These **X** signals are interpreted as linear **write** signals when the gel image is created.

Warning: A simulation experiment should not involve **Y** signals if you wish to display the output as a virtual gel image. Only **X** signals are allowed, and they are interpreted as **write** signals when a gel image is made.

To take an example, the function `example_vgel()` in `constants.py` defines a demo gel that can be displayed as a virtual gel image.

First, run a simulation of each lane of the gel, as described previously:

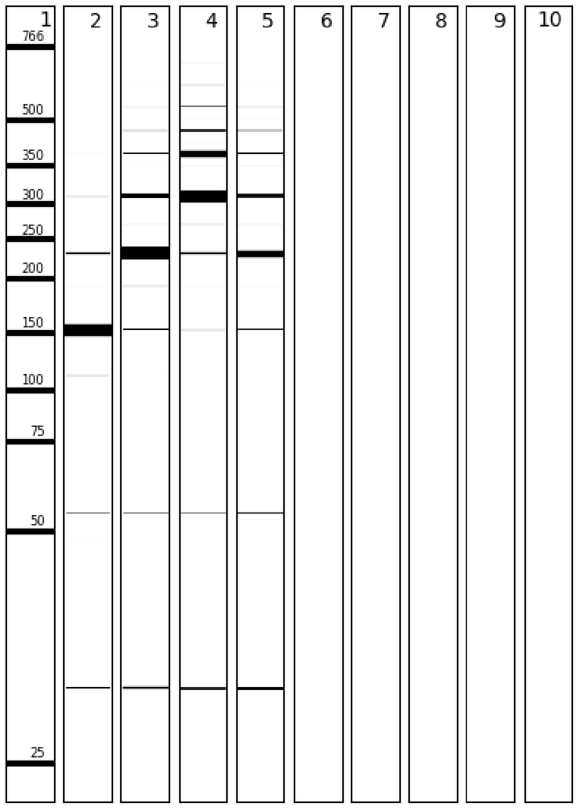
```
python3 washing.py example_vgel 2
python3 washing.py example_vgel 3
python3 washing.py example_vgel 4
python3 washing.py example_vgel 5
```

The simulation results of each lane will be stored in the `results` directory as a pickle file.

When all simulations are complete, **two alternative virtual gel images** can be made:

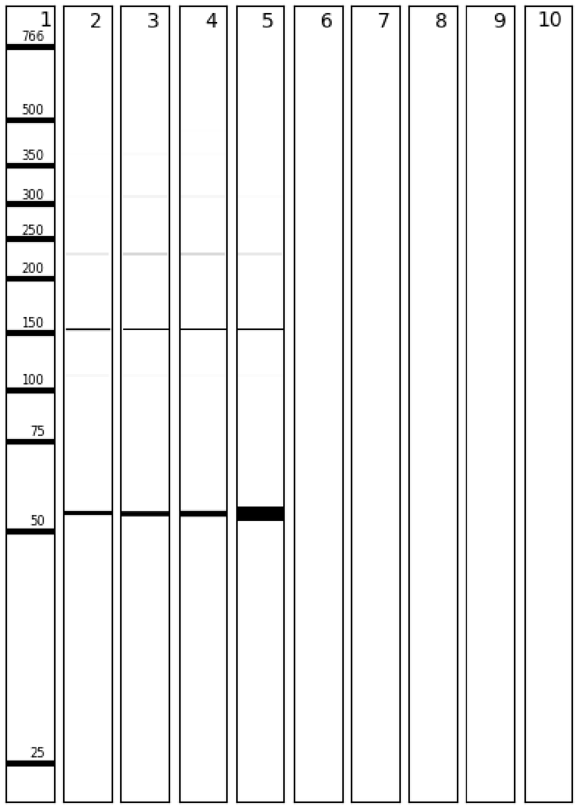
1. A virtual gel showing the **state of the supernatant solution following a final wash and addition of releaser**. The **releaser** moves stacks attached to beads into supernatant solution. This type of virtual gel is used to get an idea of system state, and is used Figure 2 of the paper:

   ```
   python3 vgel.py example_vgel releaser
   ```



2. A virtual gel showing the **state of the supernatant solution only**, after the last strand was added and the final wait time waited. Stacks attached to beads remain attached to beads and do not make it into supernatant solution. This shows the "readout" of the DNA stack device as used in Figure 4c and 4d (lanes 6,7,8) of the paper:

```
python3 vgel.py example_vgel supernatant
```



The above two commands each place a PNG image of the virtual gel in the `results` directory. The external program ImageMagick is used to rotate the images. Additionally, a text file is made, detailing the bands in each lane and their relative mass concentrations.

The NEB low molecular weight ladder is displayed in lane 1 of the virtual gel.

Band thickness is controlled by the `MASS_CONC_TO_LINEWIDTH_MULTIPLIER` parameter in `vgel.py`.

See paper Supplementary Information for electrophoretic mobility analysis of linear DNA stack complexes.