The proto-Nucleic-Acid Builder (pNAB)

1 Introduction

The proto-Nucleic Acid Builder is a software tool for constructing nucleic acid analogs. Click on the links below to learn how to install and use the program.

- Trying Online or Installing

- Examples

- Advanced Manual

Here, we present a summary of the idea of the program. For more details, refer to the published article.

1.1 Nucleic Acid Analogs

In the context of this program, Nucleic Acid Analogs refer to nucleic acids with any modifications in the nucleobase or the backbone. Additionally, the program can build hexameric nucleic acid analogs. The following image shows a few example of nucleic acid analogs that can be constructed by the program.
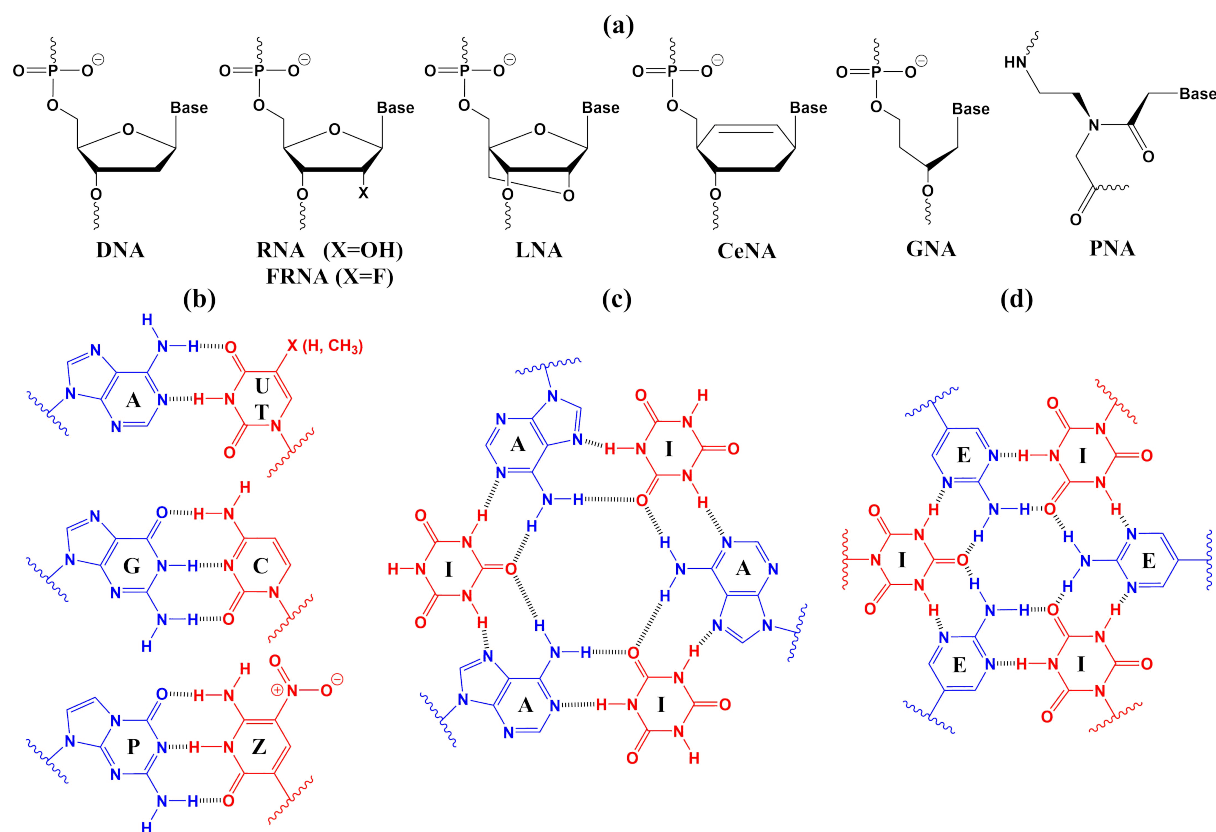
Figure 1 Chemical structures of alternative nucleic acids. Panel (a) shows examples of alternative nucleic acid backbones. Panel (b) shows the canonical nucleobases and two examples of alternative nucleobases that can be incorporated into a nucleic acid duplex. Panel (c) shows the interaction between three adenine oligomers and an alternative nucleobase. Panel (d) shows the interaction between oligomers of two alternative nucleobases and the formation of a hexmeric structure. (LNA: locked nucleic acid; CeNA: cyclohexene nucleic acid; GNA: glycol nucleic acid; PNA: peptide nucleic acid; P: 6-amino-5-nitro-2(1H)-pyridone; Z: 2-amino-imidazo[1,2-a]-1,3,5-triazin-4(8H)-one; I: cyanuric acid; E: aminopyrimidine)

1.2 Algorithm

The program builds nucleic acid according to the following algorithm:
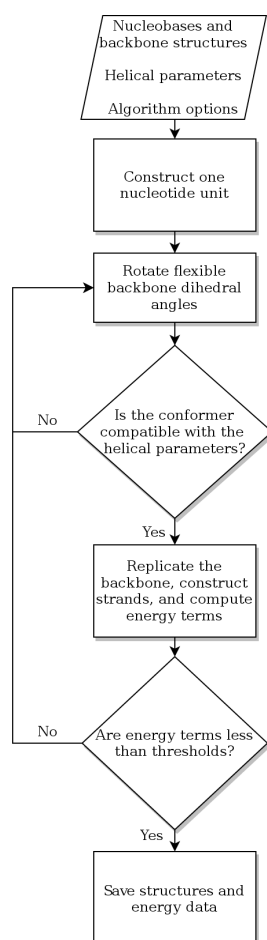
Figure 2 The algorithm for predicting alternative nucleic acid structures.

2 Trying Online or Installing

2.1 Trying the Program Online

You can explore the program online by clicking `here`. This link takes you to the graphical user interface of the program. There, you can try various available examples for building DNA, RNA, and other alternative nucleic acids. You can also explore how the structure and energy of nucleic acids change as you play with the helical parameters. Additionally, you can upload a three-dimensional structure of an alternative nucleic acid backbone or nucleobase construct a new nucleic acid analogs. See the Examples section to learn how to use the grahpical interface.

This online version is useful for testing the program. The online version uses free cloud computing resources provided by `Binder`, and this comes with several limitations. If your session remains inactive for 10-20 minutes, your session will be terminated. Moreover, the cloud computers limit how long you can remain in a session. To prevent the loss of data when the session terminates because of inactivity, the program automatically download the output files after it finishes running. You may need to adjust the settings of the browser to allow pop-up windows to open for downloading.

If you need a more permanent version of the program, you can install it in your computer.

2.2 Installation

The pNAB program is available as a conda package. Download the light-weight `miniconda` or the more fully featured `Anaconda`. Open the conda terminal and type:

```
conda create -n pnab -c alenaizan -c conda-forge pnab
conda activate pnab
```

This creates and activates a new conda environment called `pnab`.

Note

> You may need to activate the NGLView library to visualize molecules:
> ```
> jupyter-nbextension enable nglview --py --sys-prefix
> ```
>
> In Windows, the program may not compute energires correctly if the environment variable BABEL_DATADIR is not set. Set the variable to the
> ```
> share\openbabel
> ```
> folder inside your Miniconda/Anaconda folder.

2.3 Running the Program

The program can be used through a graphical user interface or a simple python script.

2.3.1 Using the Graphical User Interface

From the conda command-line, run:
```
jupyter notebook
```

Alternatively, open *Anaconda Navigator* (`details`):
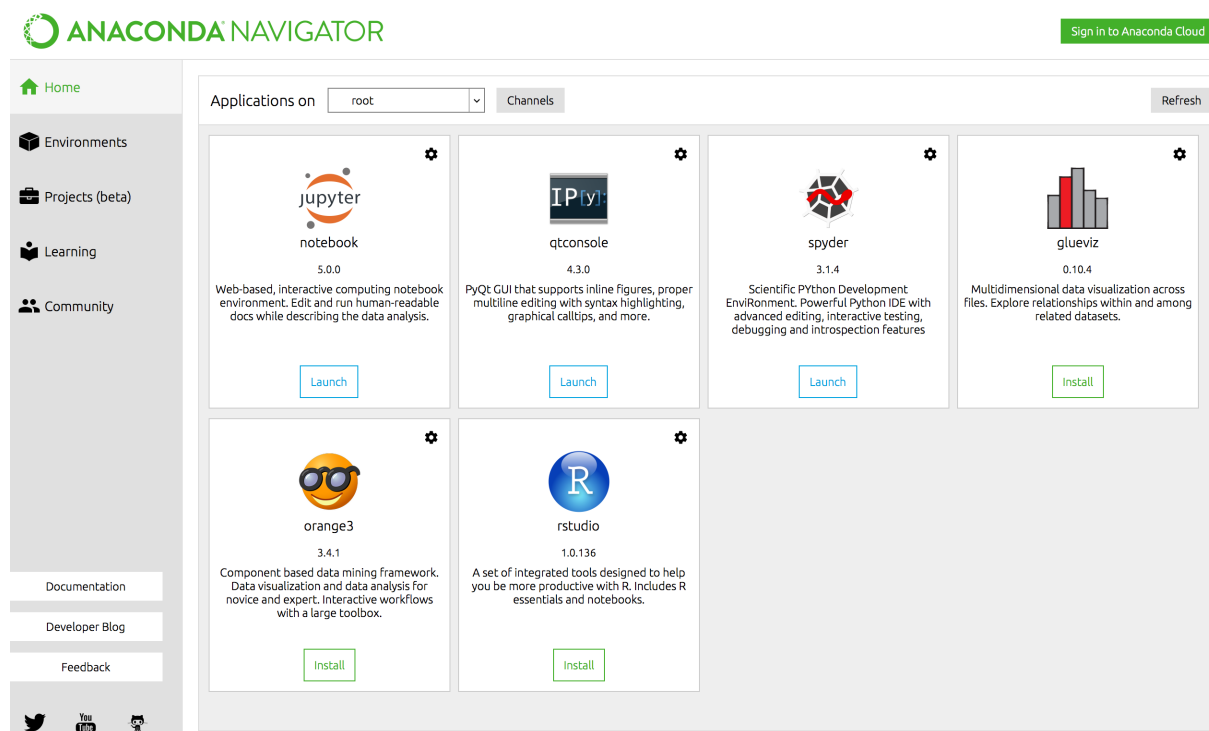
Figure 3 Anaconda Navigator

In the Jupyter Notebook tab, click on *Launch*. If a browser does not open automatically, you may need to open it this way:
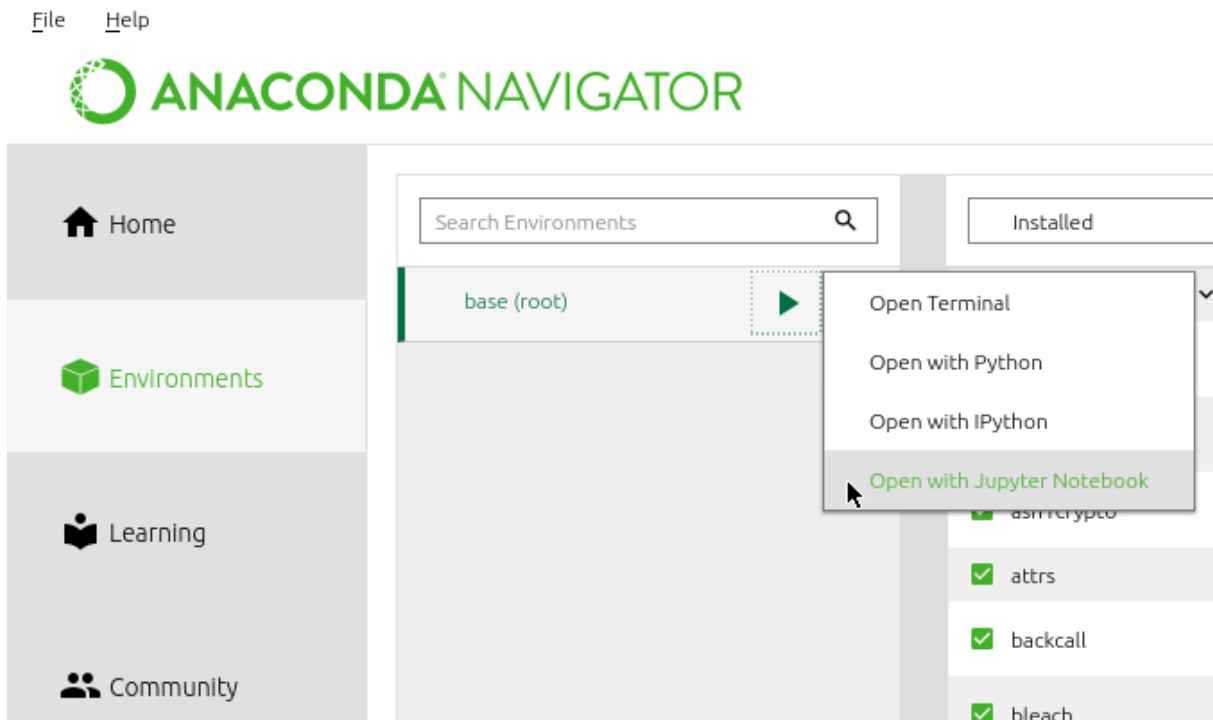
Figure 4 Launching Jupyter notebook

When a browser opens with the Jupyter page, it displays the folders in your computer. navigate to an appropriate folder or create a new folder if needed, then click on *New > python 3*
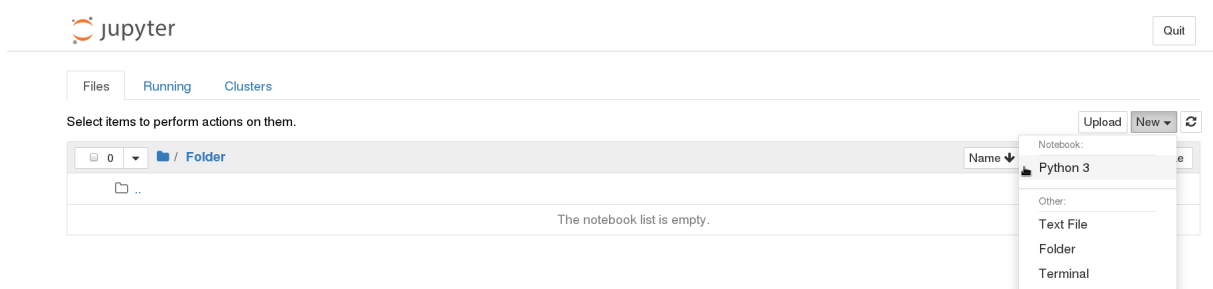
Figure 5 Creating a new Jupyter notebook

This opens an empty notebook. Type the the following two lines in the first cell and run it. This should show the graphical user interface of the program.

```
import pnab
pnab.builder()
```
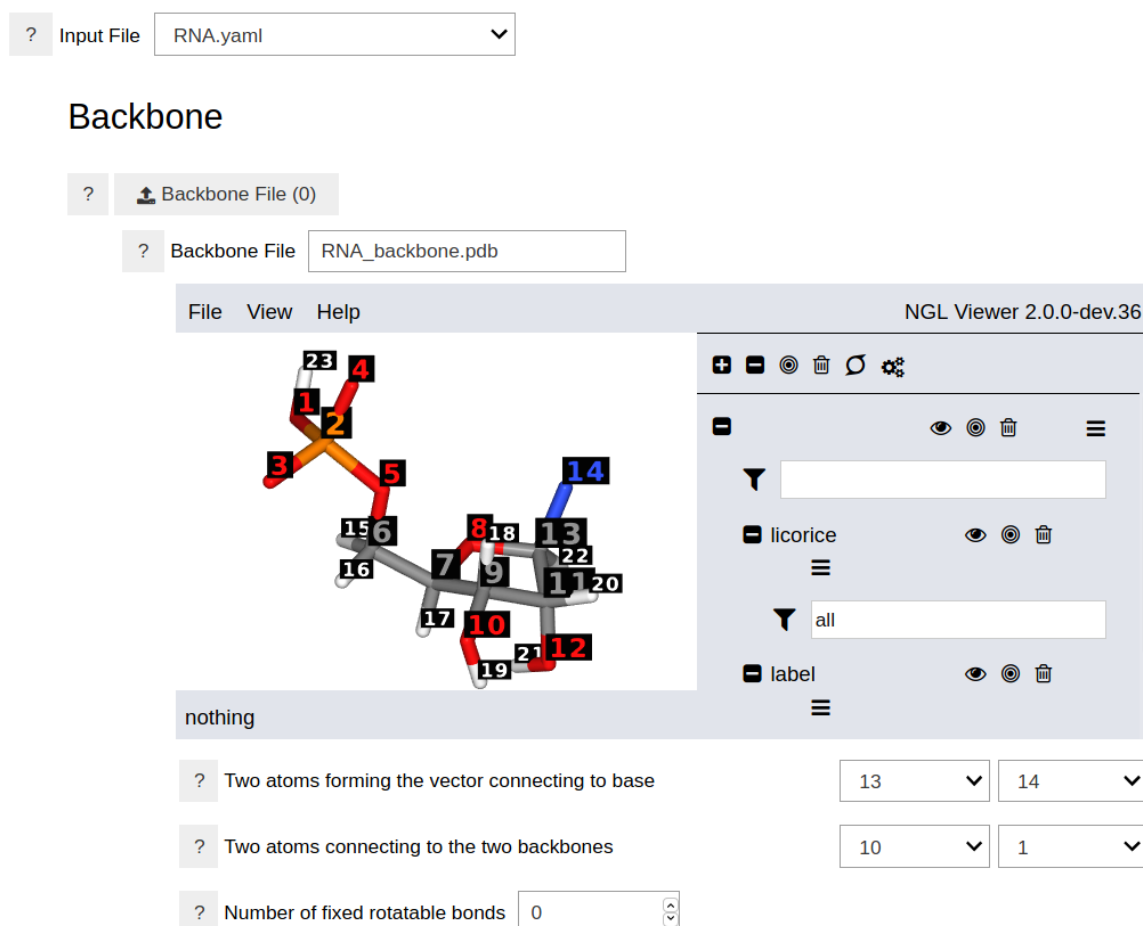
Figure 6 Running the graphical user interface

2.3.2 Using a Python Script

The following python script would run the program to generate an RNA structure. The file *RNA.yaml* is an example file available in the program.

```
import pnab
run = pnab.pNAB("RNA.yaml")
run.run()
```

To create your own input files, you can use the installed grahpical user interface or the online version. Alternatively, you can write an input file using a text editor and run it. See the Examples section to learn the content of the input files.

3   Examples

3.1   The Graphical User Interface

The graphical user interface (here) contains examples for RNA, DNA, and other nucleic acid anaolgs. The available examples generates similar structures to those published in the software article. They use a variety of nucleobases, backbones, helical parameters, and algorithm options. You can switch between these examples using the *Input File* option. You can run these examples as they are or you can use them as starting points for customizing your options. Bring the mouse pointer to the question marks ? in front of each option to get some explanation of the option.

| ? | Input File | RNA.yaml | ⌄ |

Backbone

| ? | ⬆ Backbone File (0) |

| ? | Backbone File | RNA_backbone.pdb |
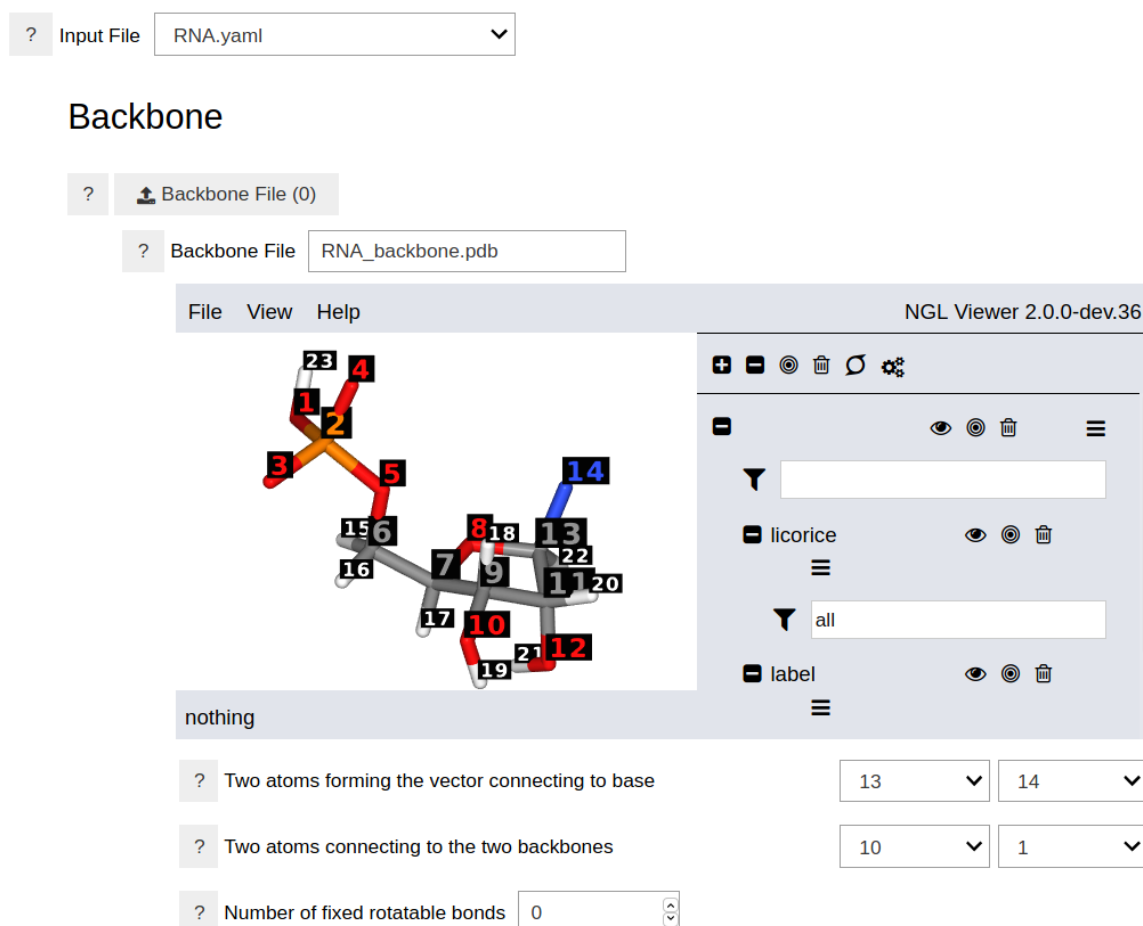
Figure 7 The graphical user interface.

After choosing or uploading the input file, the graphical interface will display the backbone and nucleobase molecules and the other options.

The options are divided into four sections:

- Backbone

- Nucleobases

- Helical Parameters

- Runtime Parameters

As shown in the above figure, specifying the backbone requires three parameters: A path to a file containing the three-dimesional structure of the backbone, two indices specifying the connection to the nucleobase, and two indices specifying the connections to the adjacent backbones. The structure of the backbone can be uploaded, and many formats (e.g. PDB) are accepted. Optional parameters include the indices of fixed rotatable bonds. This is useful when the number of rotatable bonds in the backbone are large and the search space need to be limited to specific dihedral angles.

Uploading the geometries for the nucleobases may not be necessary. The program contains a library of the canonical nucelobases: adenine (A), guaninie (G), cytosine (C), uracil (U), and thymine (T). It also contains two alternative nucleobases that can adopt the hexad geometry, namely, aminopyrimidine (E) and cyanuric acid (I). If the user wants to construct a nucleic acid analog with alternative nucleobases, then the additional nucleobases must be

defined as shown in the figure. The ZP.yaml example file shows how to define new nucleobases. The Z nucleobase is defined below.

Figure 8 Options for specifying the nucleobases.

To specify the nucleobases, a three-dimensional geometry of the nucleobases must be provided. Additionally, two indices of atoms connecting the nucleobase to the backbone must be specified. The up-to-three-letter code is used to identify the residue name in the generated PDB files. The one-letter base name is a unique name that will be used in specifying the sequence of nucleobases in the strand. It should be different from the one-letter names of the nucleobases already defined in the program library. The name of the pairing base is the one-letter code for the pairing nucleobases. It is necessary when more than one strands are constructed. The coordinates of the specified nucleobases must be in the correct reference frame. This can be done automatically by the program, but it may need to be verified by the users. The new nucleobases are aligned to the DNA/RNA frame of reference. The program cannot align the nucleobases to a hexad frame of reference.

Helical Parameters
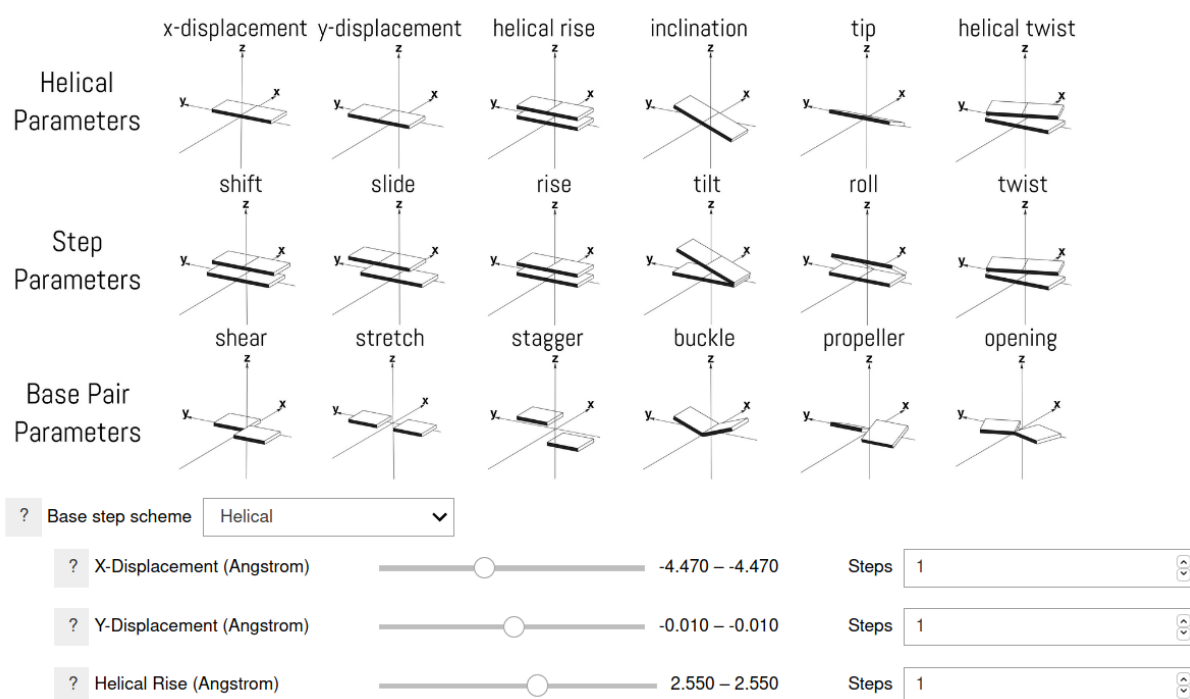
Figure 9 Specifying the helical parameters.

Six parameters specify the helical structure of the nucleic acid strands. Two equivalent schemes can be used according to whether a global helical frame or a local frame is used. Either scheme can be used for building the nucleic acid structure. Additional six parameters are used for speicifying the relative orientation of the bases in a base pair. All these options can be specified in the program. Single values or multiple values can be specified in a range. When a range of values is specified, the values are uniformally spaced between, and including, the the end points, with the specified number of configurations.

The runtime parameters include the options for specifying the conformational search algorithm and structural parameters.

Runtime Parameters

Search Algorithm

?	Search algorithm	Weighted Random Search ▾

?	The seed for the random number generator	0

?	Number of iterations to search over dihedral angles	100000000

?	Weighting temperature (K)	300

?	Quit after finding the specified number of accepted candidates	5

Distance and Energy Thresholds

?	The maximum distance between atom linkers in backbone (Angstrom)	0.8

?	Force field type	MMFF94 ▾

?	Maximum energy for newly formed bond in the backbone (kcal/mol)	0.5

?	Maximum energy for newly formed angles in the backbone (kcal/mol)	2.5

?	Maximum torsional energy for rotatable bonds (kcal/mol/nucleotide)	4

?	Maximum van der Waals energy (kcal/mol/nucleotide)	50

?	Maximum total energy (kcal/mol/nucleotide)	1000

Figure 10 Specifying the search algorithm and distance and energy thresholds.

Various conformational search algorithms are available for sapmling the dihedral angles in the backbone. Each algorithm requires specific parameters. For exmaple, the Weighted Random Search algorithm requires values for the number of conformation search steps and the temperature used in the weighting procedure. The random number seed ensures the reproducibility of the run for a given computer platform. Of the available algorithms, only the systematic search algorithm is deterministic and reproducible across platforms. Note that the conformation search procedure is also terminated after the number of candidates specified are accepted. This parameter is set to low values in the examples provided and is encouraged to be adjusted for the production runs. The distance and energy thresholds can be used to refine the generated structures and exclude unreasonable conformers. The distance threshold determines whether the conformer can adopt a periodic structure. It should be set to a small value, such as 0.8 Angstroms. The bond, angle, and torsional energies should also be set to small values. This can accelerate the search and eliminate structures with elongated bonds or strained angles. The van der Waals energy can be used to eliminate structures with significant steric clashes. However, it may vary much depending on the choice of the force field. The total energy also varies significantly with the force field.

Structural Parameters

?	FASTA string for nucleotide sequence (e.g. GCAT)	AAAAAAAAAA

? ☑ Pair A with U (Default is A-T pairing)

? ☐ Hexad strands

? Build the strand ☑   ☑   ☐   ☐   ☐   ☐

? Orientation of each strand (up or down) ☑   ☐   ☐   ☐   ☐   ☐

Run

Figure 11 Specifying the structural parameters.

The sequence of nucleobases is written as a FASTA string of the one-letter codes of the nucleobases. It needs to be written for one strand only, while the sequence of other strands is determined by the pairing nucleobase. If building a new system, it might be advisable to start with a short sequence to determine appropriate values for the helical parameters and energy thresholds. Larger strands require more time in energy computation. The default pairing scheme is to pair adenine with thymine. If an adenine-uracil base pairing is wanted, the box should be checked. The program can build hexameric proto-nucleic acid strand. If this is desired, the *Hexad strands* box should be checked. This tells the program to perform a multiple of 60-degree rotation to generate other strands. By checking on two boxes in the *Build the strand* option, the program is asked to build a duplex. Clicking on more boxes will build triplexes, quadruplexes, etc. This is appropriate only if the nucleobases are set in the hexad frame of reference, and it does not work for DNA/RNA-like structures. The strand orientation specifies whether to build parallel or anti-parallel strands. The latter is required for DNA/RNA analogs, whereas both options are possible for a hexameric nucleic acid.

The program will use the specified options for running. The time required to complete the run depends on the algorithm options. After it finishes, the program will display all the accepted candidates (if their are any) and their properties.

Conformer [1_992541.pdb ▾]

```
1_992541.pdb
Helical Parameter Set 1:  x_displacement (A): -4.47; y_displacement (A): -0.01; h_rise (A): 2.55; inclinatio
n (deg): 22.62; tip (deg): 0.07; h_twist (deg): 32.70; shear (A): -0.01; stretch (A): -0.10; stagger (A): 0.
06; buckle (deg): 0.05; propeller (deg): -10.52; opening (deg): -2.14
Distance (Angstroms): 0.455
Bond Energy (kcal/mol): 0.009
Angle Energy (kcal/mol): 2.072
Torsion Energy (kcal/mol/nucleotide): 2.175
Van der Waals Energy (kcal/mol/nucleotide): 26.058
Total Energy (kcal/mol/nucleotide): 154.186
Dihedral 1 (degrees): -159.032
Dihedral 2 (degrees): 40.585
Dihedral 3 (degrees): 170.929
Dihedral 4 (degrees): -52.280
```


Figure 12 Example results

The results are automatically downloaded as a zip file. Additionally, the output can be downloaded manually by clicking on *Download Output*. Note that downloading the output is only useful when the program is run in the cloud. When it is run locally the output files are written in the folder where the Jupyter notebook was started.

The dropdown list contains all the accepted candidates ordered by the the total energy. The output data are the distance between between one terminal atom of the backbone and the periodic image of the other terminal atom in the adjacent backbone, the energy of the new bond and the new angles formed between two nucleotides, the energy of all rotatable bonds in the backbone, the van der Waals energy, and the total energy. These six terms correspond to distance and energy thresholds defined in the input options. Additionally, the output data include the values of the dihedral angles for the rotatable bonds determined during the conformation search procedure.

3.2 Additional Examples

The graphical user interface is essentially a tool for specifying the input options and visualizing the results. However, the functionalities of the program can also be accessed through python scripts.

3.2.1 Basic Structue of the Input File

Here, we show the content of an example options file, which can be written by a text editor to specify the options. Input files can also be generated using the graphical user interface.

```
Backbone:
  file_path: RNA_backbone.pdb
  interconnects: # Backbone to backbone connection
  - 10
  - 1
  linker: # Backbone to base connection
  - 13
  - 14
```

This section specifies the backbone structure. It gives a path to the file containing the three-dimensional structure of the backbone. By default, the program first looks for a file in the current working directory. If it does not find it, it will look for it in the program library. If it does not find it there, it will raise an error. The section also gives the indices of the atoms that connect the backbone to the nucleobase as well as the indices of the atoms that connect backbones together.

An optional entry in this section is specifying fixed rotatable bonds. This can be specified as:

```
fixed_bonds:
- - 1
  - 2
- - 4
  - 10
```

This specifies that the dihedral angle specified by atom indices 1 and 2 and the dihedral angle specified by atom indices 4 and 10 will be fixed during the conformation search.

```
HelicalParameters:
  is_helical: true
  inclination:
  - 22.9 # Initial point in the range
  - 22.9 # Final point in the range
  - 1 # Number of configrations
  h_rise:
  - 2.53
  - 2.53
  - 1
  tip:
  - 0.08
  - 0.08
  - 1
  h_twist:
  - 32.39
  - 32.39
  - 1
  x_displacement:
  - -4.54
  - -4.54
  - 1
  y_displacement:
  - -0.02
  - -0.02
  - 1
```

This section specifies the helical parameters. The *is_helical* keyword determines whether the helical parameters are used or whether the step parameters (i.e. shift, slide, rise, tilt, roll, twist). These are two equivalent schemes for describing the orienation of the nucleobases. When the *is_helical* value is true, any values specified for the step parameters will not be used. The base pair parameters (shear, stretch, stagger, buckle, propeller, opening) can be similarly specified. The first value is the beginning of the range, the second value is the end of the range, and the third value is the number of configurations in the range. If you want a specific single value, simply specify the same value for both the beginning and the end of the interval.

Note that for hexad geometries, only the rise and the twist are defined. Therefore, the ranges for other entries must be set to zero or removed altogether from the input file.

```
RuntimeParameters:
  seed: 0 # Seed for the random number generator
  search_algorithm: "weighted monte carlo search" # Search algorithm
  num_steps: 10000000 # Number of steps for backbone dihedral search
  weighting_temperature: 300.0 # Temperature used in the weighting procedure for dihedral angles
  monte_carlo_temperature: 300.0 # Temperature used in the Monte Carlo procedure
  max_distance: 0.8 # Maximum distance between atom linkers in backbone
  ff_type: GAFF # Force field type
  energy_filter:
```

```
- 1.0 # Bond energy involving newly formed backbone bonds
- 2.0 # Angle energy involving newly formed backbone bonds
- 4.0 # Total torsional energy of rotatable bonds
- 0.0 # Total van der Waals energy
- 10000000000.0 # Total Energy of the system
strand: CGAUUUAGCG # Sequence of base names
build_strand: # What strands to build
- true  # Build one strand only
- false
- false
- false
- false
- false
```

This section specifies the runtime parameters for the program. Other search algorithms can be specified as follows:

```
search_algorithm: "systematic search"
dihedral_step: 2
```

```
search_algorithm: "monte carlo search"
num_steps: 10000000
monte_carlo_temperature: 300.0
```

```
search_algorithm: "random search"
num_steps: 10000000
```

```
search_algorithm: "weighted random search"
num_steps: 10000000
weighting_temperature: 300.0
```

```
search_algorithm: "genetic algorithm search"
num_steps: 1000
population_size: 10000
mutation_rate: 0.75
crossover_rate: 0.25
```

The example above shows how to build single-stranded RNA by using the keyword *build_strand* and setting the first value to true and the rest to false. To build a duplex, write:

```
build_strand:
- true
- true
- false
- false
- false
- false
strand_orientation:
- true
- false
- false
- false
- false
- false
```

For DNA and RNA, the strands are anti-parallel. The *strand_orientation* option specifies this by setting the first strand to true and the second to false.

To build a hexad,

```
is_hexad: true
build_strand:
- true
- true
- true
- true
- true
- true
strand_orientation:
- true
- true
- true
- true
- true
- true
```

The *is_hexad* keyword tells the program that we want to use the hexad geometry. The program, therefore, will perform 60 degrees rotation when appropriate to generate the strands. To build all the six strands in the hexad, set all entries in *build_strand* to true. The *strand_orientation* option allows you to build any cobmination of parallel and anti-parallel strands by setting the values to *true* or *false*

3.2.2 Adding Bases Not Defined in The Library

The program can build structures with arbitrary nucleobases. The key problem with defining a new nucleobase is making sure that the provided geometry of the nucleobase is in the correct standard frame of reference. Otherwise, the helical parameters will be ill-defined. The additional bases can be defined in the graphical user interface or by editing the input file. The program can automatically align provided nucleobases to the standard DNA/RNA frame of reference, though the user may need to verify that the nucleobase is correctly aligned. The program cannot align bases to the hexad frame of reference.

For the generated strands, the length of the bond between the nucleobases and the backbone is determined by the bond length of the first nucleotide in the strand. The bond length is determined by the van der Waals radii of the bonded atoms. It can also be set manually by the user.

The program has a library of predefined nueleobases. They include adenine, guanine, cytosine, thymine, and uracil. Additionally, there are two proto-nucleobases that are used for building the hexads, namely aminopyrimidine and cyanuric acid. To define other nucleobases, add the following entries to the input file:

```
Base modified_adenine:
  code: AD2 # Three letter PDB code
  file_path: modified_adenine.pdb # Path to the 3D structure of the nucleobase
  linker: # Indices of the vector forming the bond between the nucleobase and the backbone
  - 5
  - 11
  name: Q # One-letter name of the base
  pair_name: T # One-letter name of the pairing base
```

These lines will add the nucleobase *modified_adenine* to the library of available nucleobases. For the name of the base, you must not use any of the names already used in the library. Otherwise, your new base might be overwritten. The reserved names are the following: *A*, *G*, *C*, *T*, *U*, *I*, and *E*. These names are case-insensitive.

Now that you defined a new nucleobase, you can use its name when you specify the strand:

```
strand: CGCQUQTGQG
```

You can define additional nucleobases in the same way.

3.2.3 Adding Files to the Library

The library of the program is in the folder *data* in the package folder. The library contains example files and the coordinates for the defined bases and backbones. You can add additional files there to make them accessible wherever you run your Jupyter notebook or python script. To add other nucleobases to the library, edit the *bases↩ _library.yaml* files.

3.2.4 Python Script

The minimal input to run the program as a python script is this:
```
import pnab
run = pnab.pNAB('RNA.yaml')
run.run()
```

By default, the program first looks for the file *RNA.yaml* in the current working directory. If it does not find it, it will look for it in the program library. If it does not find it there, it will raise an error. This creates an instance of the driver.pNAB class, given the options specified in the input file, and runs the program.

To access the options programmatically, you can use the *options* attribute:
```
print(run.options)
run.options['RuntimeParameters']['search_algorithm'] = 'monte carlo search'
```

This attribute is a dictionary of all the defined options, and it can be changed in the script.

After running the program, the instance stores all the results in the *results* attribute.
```
print(run.results)
print(run.header)
print(run.prefix)
```

This is a numpy array containing the information about the accepted nucleic acid candidates, that is the candidates that satisfied the distance and energy criteria. The columns in the array are defined accroding to the corresponding entry in the *header* attribute. The first column in the results contain the prefix value. This is an integer that indicates the sequence of the helical configuration. This parameter is useful when you run a range of values for the helical parameters, where each helical configuration would have a different prefix. The *prefix* attribute is a dictionary whose keys are strings of the sequence of integers and whose values are strings specifying the corresponding helical configurations.
```
print(run.prefix['1'])
```

When multiple helical configurations are defined, e.g. by giving a range of values for the twist angle and asking for 5 configurations, the program runs these different helical configurations in parallel. The number of calculations that can be run in parallel depends on the available processors in the computer. By default, the program uses all available CPUs for parallel calculations. To use a different number, you can pass a keyword argument to the *run* function:
```
run.run(number_of_cpus=1)
```

Additionally, the program prints a progress report to the screen by default. If you do not want the progress report, type:
```
run.run(verbose=False)
```

4 Advanced Manual

4.1 Compiling and Testing the Package

Clone the GitHub repository at https://github.com/alenaizan/pnab. And compile the package. All the dependcies can be satisfied through conda.
```
conda install -c conda-forge python numpy cmake openbabel eigen pybind11 pyyaml nglview
```

The C++ compiler in linux can also be installed using Conda.
```
conda install -c conda-forge gcc_linux-64 gxx_linux-64
```

The code has been tested using the gcc 7.3 and 9.3 compilers in Linux, clang 8.0 and 10.0 compilers in Mac, and Visual Studio 2015/

The code uses the pytest package for testing. To test that the code works, first install pytest:
```
conda install pytest
```

Then, execute:
```
python -c "import pnab; pnab.test()"
```

4.2 Accessing the C++ Classes through Python

All the code for manipulating the molecule and computing the energies is written in C++. The main C++ classes that are used for defining the options can be accessed in python. It is sufficient to write an input file and call the program using this syntax:

```python
import pnab
run = pnab.pNAB('RNA.yaml')
run.run()
```

The advantage of this approach is that it performs basic validation of the user defined options. Additionally, for runs with multiple helical configurations, the independent configurations can be run in parallel using the *multiprocessing* library. This is managed internally by the python program. However, if the user wants to access the C++ classes directly through python, then this can be performed as follows:

```python
from pnab import bind
backbone = bind.Backbone()
backbone.file_path = 'rna_bb.pdb'
backbone.interconnects = [10, 1]
backbone.linker = [13, 14]
base = bind.Base()
base.file_path = 'adenine.pdb'
base.code = 'A'
base.linker = [5, 11]
base.name = 'Adenine'
base.pair_name = 'Uracil'
bases = [base]
hp = bind.HelicalParameters()
hp.h_twist = 32.39
hp.h_rise = 2.53
hp.inclination = 22.9
hp.tip = 0.08
hp.x_displacement = -4.54
hp.y_displacement = -0.02
rp = bind.RuntimeParameters()
rp.search_algorithm = 'weighted random search'
rp.num_steps = 1000000
rp.ff_type = 'GAFF'
rp.energy_filter = [10000.0, 10000.0, 10000.0, 10000.0]
rp.max_distance = 0.2
rp.strand = ['Adenine']*5
output = bind.run(rp, backbone, bases, hp, 'test')
print(output)
```

The output from the run is a string containing the data in CSV format.

The C++ classes exposed to python are only for defining options and running the program. The other classes can be accessed through the C++ code.

5 Namespace Documentation

5.1 driver Namespace Reference

This is the main file for the pnab driver.

Classes

- class pNAB

  The proto-Nucleic Acid Builder main python class.

Functions

- def init_worker ()

Variables

- • [category](#)

5.1.1 Detailed Description

This is the main file for the pnab driver.

This file contains the [pNAB](#) class for calling the C++ library.

5.1.2 Function Documentation

5.1.2.1 init_worker() `def driver.init_worker ( )`

5.1.3 Variable Documentation

5.1.3.1 category `driver.category`

5.2 jupyter_widgets Namespace Reference

A file for displaying widgets in the Jupyter notebook.

Functions

- • def [view_nglview](#) (molecule, label=False)

  Display molecules using the NGLView viewer.
- • def [fixed_bonds](#) (num_bonds, num_atoms, param)

  Display widgets for determining indices of fixed bonds.
- • def [path](#) (file_path, param)

  Display backbone to Jupyter notebook given a file path.
- • def [upload_backbone](#) (f, param)

  Upload a backbone file to Jupyter notebook.
- • def [backbone](#) (param)

  Main backbone widget for use in Jupyter notebook.
- • def [base_path](#) (file_path, param, base_number)

  Display base to Jupyter notebook given a file path.
- • def [upload_base](#) (f, param, base_number)

  Upload a base file to Jupyter notebook.
- • def [add_base](#) (number_of_bases, param)

  Display widgets to upload the requested number of bases.
- • def [bases](#) (param)

> *Bases widget for use in Jupyter notebook.*
> - def helical_parameters (param)
> - def algorithm (chosen_algorithm, param)
>
>> *Display search parameters based on the chosen algorithm.*
>
> - def runtime_parameters (param)
>
>> *Runtime parameter widget for use in Jupyter notebook.*
>
> - def upload_input (param, f)
>
>> *Widget to upload an input file.*
>
> - def display_options_widgets (param, input_file, uploaded=False)
>
>> *Display all widgets in Jupyter notebook given an input file.*
>
> - def user_input_file (param)
>
>> *Display input file options.*
>
> - def run (button)
>
>> *Function to run the code when the user finishes specifying all options.*
>
> - def extract_options ()
>
>> *Extracts user options from the widgets.*
>
> - def single_result (result, header, results, prefix)
>
>> *Interactive function to display a single result.*
>
> - def show_results (results, header, prefix)
>
>> *Display results.*
>
> - def builder ()
>
>> *The function called from the Jupyter notebook to display the widgets.*

Variables

- dictionary input_options = {}

    Stores the widgets corresponding to the user-defined options.

5.2.1 Detailed Description

A file for displaying widgets in the Jupyter notebook.

This file containts widgets for sepecifying options in the Jupyter notebook using the ipywidgets library (https://github.com/jupyter-widgets/ipywidgets). Using Jupyter notebook is not necessary for using the program, as the program can be run in as a python script. However, the widgets provide a graphical user interface for making input files, running the code, and displaying the results. The Jupyter notebook can be run locally or on the clouds using Binder (https://mybinder.← org/v2/gh/alenaizan/pnab/master?filepath=binder).

The widgets have four main components: Backbone, Bases, Helical Parameters, and Runtime Parameters. The Bases section does not require the user input but displays some information about the available nucleobases in the library. The NGLView library is used for visualization

To run the widgets from a Jupyter notebook, simply execute the following:
```
import pnab
pnab.builder()
```

5.2.2 Function Documentation

5.2.2.1 add_base() def jupyter_widgets.add_base (
                number_of_bases,
                param)

Display widgets to upload the requested number of bases.

Parameters

number_of_bases	(int) The number of additional bases to define
param	(dict) *options._options_dict*['Base'] and dictionaries for the other defined bases

See also

> bases
>
> upload_base
>
> options._options_dict
>
> view_nglview

5.2.2.2 algorithm() `def jupyter_widgets.algorithm (`
> `chosen_algorithm,`
> `param )`

Display search parameters based on the chosen algorithm.

There are six search algorithms and each one has a set of input parameters. The search algorithms are:

- Systematic Search: Requires specifying the dihedral angle step size, *dihedral_step*.

- Monte Carlo Search: Requires specifying the number of steps, *num_steps*, and the Monte Carlo temperature, *monte_carlo_temperature*.

- Weighted Monte Carlo Search: Also requires specifying the weighting temperature, *weighting_temperature*.

- Random Search: Requires specifying the number of steps, *num_steps*.

- Weighted Random Search: Also requires specifying the weighting temperature, *weighting_temperature*.

- Genetic Algorithm Search: Requires specifying the number of generations, *num_steps*, the population size, *population_size*, the mutation_rate, *mutation_rate*, and the crossover rate, *crossover_rate*.

Parameters

chosen_algorithm	(str) The chosen algorithm
param	(dict) *options._options_dict*['RuntimeParameters']

Returns

> None; *jupyter_widgets.input_options* is modified in place

See also

> runtime_parameters
>
> input_options
>
> options._options_dict

5.2.2.3 backbone() `def jupyter_widgets.backbone (`
             `param` `)`

Main backbone widget for use in Jupyter notebook.

This function displays the backbone widgets.

Parameters

param	(dict) *options._options_dict*['Backbone']

Returns

    None

See also

    upload_backbone

    options._options_dict

    display_options_widgets

5.2.2.4 base_path() `def jupyter_widgets.base_path (`
             `file_path,`
             `param,`
             `base_number` `)`

Display base to Jupyter notebook given a file path.

This function displays the options for the base. If the *file_path* does not exist, this function displays nothing.

Parameters

file_path	(str) Path to a file containing the 3D structure of the base molecule
param	(dict) *options._options_dict*['Base']
base_number	The number of the new base

Returns

    None; *jupyter_widgets.input_options* is modified in place

See also

    upload_base

    input_options

    options._options_dict

    view_nglview

5.2.2.5 bases() `def jupyter_widgets.bases (`
           param `)`

Bases widget for use in Jupyter notebook.

This function displays information on the available nucleobases that are defined in the program. The bases are defined in "data/bases_library.yaml"

Returns

    None

See also

    display_options_widgets

5.2.2.6 builder() `def jupyter_widgets.builder ( )`

The function called from the Jupyter notebook to display the widgets.

Execute the following to display all the widgets in the notebook

```
import pnab
pnab.builder()
```

Returns

    None

5.2.2.7 display_options_widgets() `def jupyter_widgets.display_options_widgets (`
           param,
           input_file,
           uploaded = False `)`

Display all widgets in Jupyter notebook given an input file.

If *input_file* is provided, then it updates the default options in *options._options_dict* to display the user-defined options. If *input_file* is "Upload file", then an ipywidgets.FileUpload widget is displayed. This function also displays a widget for running the program after the user defines all the options.

Parameters

param	(dict) *options._options_dict*
input_file	(str) Input file
uploaded	(bool) Whether a file is uploaded by the user or not

Returns

    None

See also

    backbone

    bases

    helical_parameters

    runtime_parameters

    upload_input

    run

    user_input_file

    options._options_dict

5.2.2.8   extract_options()   `def jupyter_widgets.extract_options ( )`

Extracts user options from the widgets.

This function extracts the values of all options specifed by the user.   It access all the widgets in *jupyter_widgets.input_options* and extracts their values.

Returns

    user_options A dictionary of the values of all the user-defined options

See also

    run

    input_options

    options._options_dict

5.2.2.9   fixed_bonds()   `def jupyter_widgets.fixed_bonds (`
           `num_bonds,`
           `num_atoms,`
           `param )`

Display widgets for determining indices of fixed bonds.

This function is changed interactively bases on the number of fixed bonds requested by the user.

Parameters

num_bonds	(int) number of fixed bonds; changed dynamically by the user
num_atoms	(int) number of atoms in the backbone
param	(dict) *options._options_dict*['Backbone']

Returns

None; *jupyter_widgets.input_options* is modified in place

See also

path

input_options

options._options_dict

5.2.2.10 helical_parameters() `def jupyter_widgets.helical_parameters (`
      param `)`

Display widgets for specifying helical parameters. It also displays an image illustrating the used helical parameters.

```
@param param (dict) @a options._options_dict['HelicalParameters']

@returns None; @a jupyter_widgets.input_options is modified in place

@sa input_options
@sa display_options_widgets
@sa options._options_dict
```

5.2.2.11 path() `def jupyter_widgets.path (`
      file_path,
      param `)`

Display backbone to Jupyter notebook given a file path.

This function displays the options for the backbone. If the *file_path* does not exist, this function displays nothing.

Parameters

file_path	(str) Path to a file containing the 3D structure of a backbone molecule
param	(dict) *options._options_dict*['Backbone']

Returns

None; *jupyter_widgets.input_options* is modified in place

See also

upload_backbone

input_options

options._options_dict

view_nglview

5.2.2.12 run() `def jupyter_widgets.run (`
　　　　　　　　`button )`

Function to run the code when the user finishes specifying all options.

It extracts the user-defined options and creates a *pNAB.pNAB* instance. Then, it run the code and display the results.

Parameters

button	(variable) variable for using the widgets.Button.on_click method

See also

> [display_options_widgets](#)
>
> pNAB.pNAB
>
> [show_results](#)

5.2.2.13 runtime_parameters() `def jupyter_widgets.runtime_parameters (`
　　　　　　　　`param )`

Runtime parameter widget for use in Jupyter notebook.

Displays widgets for specifying runtime parameters.

Parameters

param	(dict) [options._options_dict](#)['RuntimeParameters']

Returns

> None; [jupyter_widgets.input_options](#) is modified in place

See also

> [algorithm](#)
>
> [input_options](#)
>
> [display_options_widgets](#)
>
> [options._options_dict](#)

5.2.2.14 show_results() `def jupyter_widgets.show_results (`
　　　　　　　　`results,`
　　　　　　　　`header,`
　　　　　　　　`prefix )`

Display results.

A function to display all the accepted candidates using a dropdown list.

Parameters

results	(np.ndarray) numpy array of results sorted by total energy
header	(str) A comma separated string of the output properties of the conformer
prefix	(dict) A dictionary of the prefix of the run and the associated helical configuration

Returns

> None

See also

> run
>
> single_result

5.2.2.15 single_result() `def jupyter_widgets.single_result (`
> `result,`
> `header,`
> `results,`
> `prefix )`

Interactive function to display a single result.

It displays one accepted conformer and prints its energies and other properties.

Parameters

result	(int) index of the conformer in the *results* array
header	(str) A comma separated string of the output properties of the conformer
results	(np.ndarray) a numpy array of all the accepted conformers
prefix	(dict) A dictionary of the prefix of the run and the associated helical configuration

Returns

> None

See also

> show_results

5.2.2.16 upload_backbone() `def jupyter_widgets.upload_backbone (`
> `f,`
> `param )`

Upload a backbone file to Jupyter notebook.

This function is used to upload a backbone file and write it. If a file is already specified, then the backbone is displayed. If a file is uploaded, this function writes the file to the current working directory. Then, the function calls the *jupyter_widgets.path* function interactively with the backbone file path.

Parameters

f	(ipywidgets.FileUpload) File upload widget
param	(dict) *options._options_dict*['Backbone']

Returns

> None

See also

> path
>
> backbone

5.2.2.17 upload_base() `def jupyter_widgets.upload_base (`
> `f,`
> `param,`
> `base_number )`

Upload a base file to Jupyter notebook.

This function is used to upload a base file and write it. If a file is already specified, then the base is displayed. If a file is uploaded, this function writes the file to the current working directory. Then, the function calls the *jupyter_widgets.base_path* function interactively with the base file path.

Parameters

f	(ipywidgets.FileUpload) File upload widget
param	(dict) *options._options_dict*['Base'] for the appropriate base number
base_number	(int) The number of the additional base

Returns

> None

See also

> base_path
>
> add_base

5.2.2.18 upload_input() `def jupyter_widgets.upload_input (`
> `param,`
> `f )`

Widget to upload an input file.

If a file is uploaded, it is written in the current working directory. Then, this function calls *jupyter_widgets.display_options_widgets* function with the newly written file.

Parameters

param	(dict) *options._options_dict*
f	(ipywidgets.FileUpload) file upload widget

Returns

> None

See also

> display_options_widgets

5.2.2.19 user_input_file() `def jupyter_widgets.user_input_file (`
> *param* `)`

Display input file options.

Gives the user several input files that they can try, and the option that they upload their input file. Once an input file is chosen, all widgets are displayed with the user-defined options.

Parameters

param	(dict) *options._options_dict*

Returns

> None

See also

> display_options_widgets

5.2.2.20 view_nglview() `def jupyter_widgets.view_nglview (`
> *molecule,*
> *label = False* `)`

Display molecules using the NGLView viewer.

A function to view molecules using the NGLView project (https://github.com/arose/nglview). It is used to display the geometry of the backbone with atom index labels. It is also used to display accepted nucleic acid candidates generated after the search. For backbone molecules, the number of atoms in the backbone is computed here and returned to be used for bounding the displayed atomic indices in backbone widgets.

Parameters

molecule	(str) Path to a file containing the 3D structure of the molecule
label	(bool) Whether to display atom index labels

Returns

number of atoms in *molecule* if *label* is True, else None

Attention

If the molecules are not displayed correctly, you may need to execute
`jupyter-nbextension enable nglview --py --sys-prefix`

See also

[jupyter_widgets.path](#)

[jupyter_widgets.single_result](#)

5.2.3 Variable Documentation

5.2.3.1 input_options `dictionary jupyter_widgets.input_options = {}`

Stores the widgets corresponding to the user-defined options.

This dictionary has the same structure as the [options._options_dict](#) dictionary that has all the available options for the code. The widgets are added to this dictionary and the values of the widgets are extracted when the user runs the code

See also

[options._options_dict](#)

[extract_options](#)

5.3 options Namespace Reference

A file for preparing, explaining and validating options.

Functions

- def [validate_all_options](options)

    A method to validate all options.
- def [_align_nucleobase](base_options)

    Aligns the provided nucleobase to purine or pyrimidine in the nucleic acid base pair standard reference frame.
- def [_validate_input_file](file_name)

    Method to validate that the given file exists.
- def [_validate_atom_indices](indices)

    Method to validate provided lists of atom indices.
- def [_validate_helical_parameters](hp_i)

    Method to validate provided helical parameters for each parameter.
- def [_validate_energy_filter](energy_filter)

    Method to validate provided energy filter.
- def [_validate_strand](strand)

    Method to validate provided strand sequence.
- def [_validate_bool_list](x)

    Method to validate a list of bools.

Variables

- dictionary [_options_dict](= {})

    Nucleic acid builder options used in the driver.

5.3.1 Detailed Description

A file for preparing, explaining and validating options.

This option file defines a dictionary containing all the available options for the builder code. *options._options_dict* has three keys: Backbone, HelicalParameters, and RuntimeParameters. Each of these contains a dictionary of available options in each category. Each dictionary contains a short glossory that describe the option, a long glossory giving hints on how to use this option, a default value, and a validation scheme.

This file also contains additional functions to help in the validation.

5.3.2 Function Documentation

5.3.2.1 _align_nucleobase() `def options._align_nucleobase (`
    base_options) `[private]`

Aligns the provided nucleobase to purine or pyrimidine in the nucleic acid base pair standard reference frame.

This function aligns the provided nucleobase to either guanine if it has two rings or to uracil if it has one ring. The provided linker atoms that connect the base to the backbone are used for alignment. A third atom used in the alignment is determined by rearraning the atoms in the molecules using the canonical order. The function loops over all atoms in the provided nucleobase and the reference nucleobase and if two atoms have the same atomic number then these atoms are used for the alignment. A new file with the aligned nucleobase is created and used instead of the provided file.

Parameters

base_options	(dict) The nucleobase options defined in the input file

5.3.2.2 _validate_atom_indices() `def options._validate_atom_indices (`
     `indices ) [private]`

Method to validate provided lists of atom indices.

Check whether the list has two atom indices

Parameters

indices	(str|list) atom indices

Returns

> indices after validation

See also

> validate_all_options
>
> _options_dict

5.3.2.3 _validate_bool_list() `def options._validate_bool_list (`
     `x ) [private]`

Method to validate a list of bools.

Parameters

x	(list) list of bools

Returns

> validated list of bools

See also

> validate_all_options
>
> _options_dict

5.3.2.4 _validate_energy_filter() `def options._validate_energy_filter (`
            `energy_filter )  [private]`

Method to validate provided energy filter.

Energy filter has five thresholds [bond, angle, torsion, van der Waals, total]

Parameters

energy_filter	(str|list) A list of five energy thresholds

Returns

energy_filter after validation

See also

validate_all_options

_options_dict

5.3.2.5 _validate_helical_parameters() `def options._validate_helical_parameters (`
            `hp_i )  [private]`

Method to validate provided helical parameters for each parameter.

Check whether the list has correct helical parameter specifications. The correct specifications are [initial value in a range, final value in a range, number of steps]. If a single value is provided, then we assume it is the only value in a range. if two values are provided, then we assume it is a range with one configuration.

This specification is for internal use in the driver for generating multiple configurations and running them in parallel. The C++ code accepts a single value for each helical parameter.

Parameters

hp↩ _i	(str|float|list) Specifications for helical parameter i

Returns

hp_i after validation

See also

validate_all_options

_options_dict

pNAB.pNAB.run

5.3.2.6 _validate_input_file() `def options._validate_input_file (`
    `file_name` `)` `[private]`

Method to validate that the given file exists.

Check whether the file exists or not. If it does not exist as it is, the function checks whether the file is in the "pnab/data" directory.

Parameters

file_name	(str) Path to a file

Returns

 file_name after validation

See also

 validate_all_options

 _options_dict

5.3.2.7 _validate_strand() `def options._validate_strand (`
    `strand` `)` `[private]`

Method to validate provided strand sequence.

We use FASTA strings to specify the sequence in the strand. This is for use in the driver. The C++ code accepts base names with more than one letter. However, writing a FASTA sequence is easier.

The names of the bases are converted to upper case letters

Parameters

strand	(str|list) FASTA sequence of the strand

Returns

 strand list after validation

See also

 validate_all_options

 _options_dict

5.3.2.8 validate_all_options() `def options.validate_all_options (`

               options `)`

A method to validate all options.

It loops over all available builder options and validates the user input. For options that have not been specified by the user, a default value is provided.

Parameters

options	(dict) A dictionary containing all the user-defined options

Returns

    None; *options* is modified in place

See also

    pNAB.pNAB.__init__

    _options_dict

5.3.3 Variable Documentation

5.3.3.1 _options_dict `dictionary options._options_dict = {} [private]`

Nucleic acid builder options used in the driver.

This is an internal options dictionary to be used for listing, explaining and validating the options used in the code. All options here have corresponding options in the C++ code, except *options._options_dict*['Runtime⏎ Parameters']['pair_a_u'], which is used in the driver to override the default adenine-thymine pairing.

See also

    jupyter_widgets.builder()

    PNAB::Backbone

    PNAB::HelicalParameters

    PNAB::RuntimeParameters

5.4 PNAB Namespace Reference

The PNAB name space contains all the C++ classes and functions for the proto-Nucleic Acid Builder.

Classes

- class [Backbone]

  Class for holding backbone information.
- class [Base]

  Class to fully define bases (i.e. Adenine, Cytosine)
- class [Bases]

  A class that contains a vector of all the defined bases and a funtion to return a base and the complimentary base for all the bases defined.
- class [BaseUnit]

  Class to hold bases with backbones attached (nucleotides), along with associated necessary information.
- class [Chain]

  A class for building nucleic acid strands and evaluating their energies.
- class [ConformationSearch]

  A rotor search function used to find acceptable conformations of arbitrary backbone and helical parameter combinations. The main class of the proto-Nucleic Acid Builder.
- struct [ConformerData]

  Class to contain important information for an individual conformer.
- class [HelicalParameters]

  A class for holding values for all helical parameters.
- class [RuntimeParameters]

  A class for holding necessary and optional runtime parameters for conformational searches.

Functions

- std::string [run] ([PNAB::RuntimeParameters] runtime_params, [PNAB::Backbone] &py_backbone, std::vector< [PNAB::Base] > py_bases, [PNAB::HelicalParameters] hp, std::string prefix="run", bool verbose=true)

  A wrapper function to run the search algorithm code from python.
- [PYBIND11_MODULE] (bind, m)

  Exports certain classes to python to allow the user to run the code from python.

5.4.1 Detailed Description

The [PNAB] name space contains all the C++ classes and functions for the proto-Nucleic Acid Builder.

5.4.2 Function Documentation

5.4.2.1 PYBIND11_MODULE() PNAB::PYBIND11_MODULE (

```
        bind ,
        m  )
```

Exports certain classes to python to allow the user to run the code from python.

This pybind11 scheme exports only the input runtime, helical, base, and backbone parameters. It exports a single run funtion that can be called from python to run the code.

See also

[RuntimeParameters]

[HelicalParameters]

[Base]

[Backbone]

[run]

5.4.2.2 run() `std::string PNAB::run (`

           `PNAB::RuntimeParameters` *runtime_params,*

           `PNAB::Backbone &` *py_backbone,*

           `std::vector< PNAB::Base >` *py_bases,*

           `PNAB::HelicalParameters` *hp,*

           `std::string` *prefix = "run",*

           `bool` *verbose = true)*

A wrapper function to run the search algorithm code from python.

Parameters

runtime_params	The runtime parameters defined in the python script
py_backbone	The backbone defined in the python script
py_bases	A vector of the bases defined in the python script
hp	The helical parameters defined in the python script
prefix	A string the prepends the names of the output PDB files, default to "run"
verbose	Whether to print progress report to screen, default to true

Returns

      A CSV string containing the properties of the accepted candidates

6 Class Documentation

6.1 PNAB::Backbone Class Reference

Class for holding backbone information.

`#include <Containers.h>`

Public Member Functions

- Backbone ()

    Empty constructor.
- Backbone (std::string file_path, std::array< unsigned, 2 > interconnects, std::array< unsigned, 2 > linker, std::vector< std::vector< unsigned >> fixed_bonds={})

    Constructor for the backbone unit.
- OpenBabel::OBAtom ∗ getHead ()

    Gives the pointer to an atom that is the head from Backbone::interconnects{head, tail}.
- OpenBabel::OBAtom ∗ getTail ()

    Gives the pointer to an atom that is the tail from Backbone::interconnects{head, tail}.
- OpenBabel::OBAtom ∗ getLinker ()

    Get the first Backbone::linker atom pointer.
- OpenBabel::OBAtom ∗ getVector ()

    Get the second Backbone::linker atom pointer (which is probably a hydrogen)
- void center ()

    Centers the molecule. Basically just an alias of the Center() function from OpenBabel.
- void rotate (double ∗rot)

Rotates the molecule by a matrix. Basically just an alias of the Rotate() function from OpenBabel.

- void translate (OpenBabel::vector3 vec)

    Translates the molecule by a vector. Basically just an alias of the Translate() function from OpenBabel.

- OpenBabel::OBMol getMolecule ()

    Gives a copy of the molecule in the backbone, Backbone::backbone.

- void deleteVectorAtom ()

    Deletes the atom from getVector() safely. If the atom is already deleted, nothing happens.

Public Attributes

- std::array< unsigned, 2 > interconnects

    The atom indices that define the periodic conditions between backbones { head, tail }.

- std::array< unsigned, 2 > linker

    The atom indices used to align and connect backbone to base in the nucleotide.

- std::vector< std::vector< unsigned > > fixed_bonds

    A vector containing pairs of indices defining fixed rotatable bonds during dihedral search.

- OpenBabel::OBMol backbone

    The molecule for the backbone.

- std::string file_path

    The path to the file containing the molecule.

Private Member Functions

- void validate ()

    Does some basic sanity checks (such as whether or not the indices of the atom are within the range of the molecule)

Private Attributes

- bool vector_atom_deleted

    Whether or not the atom from getVector() has been deleted.

6.1.1   Detailed Description

Class for holding backbone information.

The backbone here refers to the backbone in a single nucleotide. This class holds information on the molecular structure of the backbone and the bonds that the backbone form with the nucleobases and the adjacent backbones. This class also has functions to manipulate the backbone.

See also

> BaseUnit
>
> ConformationSearch

6.1.2   Constructor & Destructor Documentation

6.1.2.1 Backbone() **[1/2]** `PNAB::Backbone::Backbone ( )` `[inline]`

Empty constructor.

This empty constructor can be used. After that, values for the member variables should be specified.

6.1.2.2 Backbone() **[2/2]** `Backbone::Backbone (`
`        std::string file_path,`
`        std::array< unsigned, 2 > interconnects,`
`        std::array< unsigned, 2 > linker,`
`        std::vector< std::vector< unsigned >> fixed_bonds = {} )`

Constructor for the backbone unit.

Parameters

file_path	The path to the file containing the molecule.
interconnects	The atom indices that define the periodic conditions between backbones { head, tail }.
linker	The atom indices used to align and connect backbone to base in the nucleotide.
fixed_bonds	A vector containing pairs of indices defining fixed rotatable bonds during dihedral search.

6.1.3 Member Function Documentation

6.1.3.1 center() `void PNAB::Backbone::center ( )` `[inline]`

Centers the molecule. Basically just an alias of the Center() function from OpenBabel.

6.1.3.2 deleteVectorAtom() `void Backbone::deleteVectorAtom ( )`

Deletes the atom from getVector() safely. If the atom is already deleted, nothing happens.

6.1.3.3 getHead() `OpenBabel::OBAtom* PNAB::Backbone::getHead ( )` `[inline]`

Gives the pointer to an atom that is the head from Backbone::interconnects{head, tail}.

Returns

The atom pointer from the backbone OBMol object

6.1.3.4 getLinker() `OpenBabel::OBAtom* PNAB::Backbone::getLinker ( )` `[inline]`

Get the first Backbone::linker atom pointer.

Returns

Pointer to the atom that is the one linking to the Base

6.1.3.5 getMolecule() `OpenBabel::OBMol PNAB::Backbone::getMolecule ( )` `[inline]`

Gives a copy of the molecule in the backbone, Backbone::backbone.

Returns

A copy of the backbone molecule

6.1.3.6 getTail() `OpenBabel::OBAtom* PNAB::Backbone::getTail ( )` `[inline]`

Gives the pointer to an atom that is the tail from Backbone::interconnects{head, tail}.

Returns

Pointer to the atom for the tail

6.1.3.7 getVector() `OpenBabel::OBAtom* PNAB::Backbone::getVector ( )` `[inline]`

Get the second Backbone::linker atom pointer (which is probably a hydrogen)

Returns

Pointer to the atom that defines the vector from the backbone to the base

6.1.3.8 rotate() `void PNAB::Backbone::rotate (`
            `double * rot )` `[inline]`

Rotates the molecule by a matrix. Basically just an alias of the Rotate() function from OpenBabel.

Parameters

rot	The matrix by which to rotate the molecule

6.1.3.9 translate() `void PNAB::Backbone::translate (`
            `OpenBabel::vector3 vec )  [inline]`

Translates the molecule by a vector. Basically just an alias of the Translate() function from OpenBabel.

Parameters

vec	The vector by which to translate the molecule

6.1.3.10 validate() `void Backbone::validate ( )  [private]`

Does some basic sanity checks (such as whether or not the indices of the atom are within the range of the molecule)

6.1.4 Member Data Documentation

6.1.4.1 backbone `OpenBabel::OBMol PNAB::Backbone::backbone`

The molecule for the backbone.

6.1.4.2 file_path `std::string PNAB::Backbone::file_path`

The path to the file containing the molecule.

6.1.4.3 fixed_bonds `std::vector<std::vector<unsigned> > PNAB::Backbone::fixed_bonds`

A vector containing pairs of indices defining fixed rotatable bonds during dihedral search.

6.1.4.4 interconnects `std::array<unsigned , 2> PNAB::Backbone::interconnects`

The atom indices that define the periodic conditions between backbones { head, tail }.

6.1.4.5 linker `std::array<unsigned , 2> PNAB::Backbone::linker`

The atom indices used to align and connect backbone to base in the nucleotide.

6.1.4.6 vector_atom_deleted `bool PNAB::Backbone::vector_atom_deleted [private]`

Whether or not the atom from getVector() has been deleted.

The documentation for this class was generated from the following files:

- Containers.h
- Containers.cpp

6.2 PNAB::Base Class Reference

Class to fully define bases (i.e. Adenine, Cytosine)

```
#include <Containers.h>
```

Public Member Functions

- Base ()

    Empty constructor.
- Base (std::string name, std::string code, std::string file_path, std::array< std::size_t, 2 > linker, std::string pair_name="")

    Create Base from basic set of parameters.
- OpenBabel::OBAtom ∗ getLinker ()

    Gives the atom of the base that connects directly to the backbone, Base::linker[0].
- OpenBabel::OBAtom ∗ getVector ()

    Gives the (most likely hydrogen) atom of the base connected to the atom from getLinker() which defines how the base connects, Base::linker[1].
- OpenBabel::OBMol getMolecule ()

    Returns a copy of the base molecule, Base::base.
- std::string getCode ()

    Gives the three-letter code of the base, Base::code.
- std::string getName ()

    Gives the full name of the base, Base::name.
- void deleteVectorAtom ()

    Deletes the atom from getVector() safely. If the atom is already deleted, nothing happens.
- std::string getBasePairName ()

    Get the name of the pair base, Base::pair_name.

Public Attributes

- std::string name

    Full name of base (i.e. "Adenine" or just "A")

- std::string code

    Three character code to define base ("Adenine": "ADE")

- std::string pair_name

    Name of the pair base.

- std::string file_path

    Path to a file containing the base.

- OpenBabel::OBMol base

    The OBMol defining the base.

- std::array< std::size_t, 2 > linker

    Holds indices for atoms forming a vector to connect to backbone {linker, hydrogen}.

Private Member Functions

- void validate ()

    Does some basic sanity checks (such as whether or not the indices of the atom are within the range of the molecule).

Private Attributes

- bool vector_atom_deleted

    Whether or not the getVector() atom was deleted.

6.2.1   Detailed Description

Class to fully define bases (i.e. Adenine, Cytosine)

This class holds information on the molecular structure of one nucleobase and the bond that it should form with the backbone. This class also has functions to manipulate the nucleobase and gets information about it.

See also

> Bases
>
> BaseUnit

6.2.2   Constructor & Destructor Documentation

6.2.2.1   Base() **[1/2]**   `PNAB::Base::Base ( )  [inline]`

Empty constructor.

This empty constructor can be used. After that, values for the member variables should be specified.

6.2.2.2   Base() **[2/2]**   `Base::Base (`
```
            std::string name,
            std::string code,
            std::string file_path,
            std::array< std::size_t, 2 > linker,
            std::string pair_name = "" )
```

Create Base from basic set of parameters.

Parameters

name	name of the base
code	three-letter code
file_path	path to the backbone file
linker	indices for atoms forming the vector connecting to the backbone
pair_name	Name of the pairing base

6.2.3 Member Function Documentation

6.2.3.1 deleteVectorAtom() `void PNAB::Base::deleteVectorAtom ( ) [inline]`

Deletes the atom from getVector() safely. If the atom is already deleted, nothing happens.

6.2.3.2 getBasePairName() `std::string PNAB::Base::getBasePairName ( ) [inline]`

Get the name of the pair base, Base::pair_name.

Returns

Pair name

6.2.3.3 getCode() `std::string PNAB::Base::getCode ( ) [inline]`

Gives the three-letter code of the base, Base::code.

Returns

The code of the base

6.2.3.4 getLinker() `OpenBabel::OBAtom* PNAB::Base::getLinker ( ) [inline]`

Gives the atom of the base that connects directly to the backbone, Base::linker[0].

Returns

A pointer to the atom that connects to the backbone

6.2.3.5   getMolecule()   `OpenBabel::OBMol PNAB::Base::getMolecule ( )  [inline]`

Returns a copy of the base molecule, Base::base.

Returns

A copy of the base molecule

6.2.3.6   getName()   `std::string PNAB::Base::getName ( )  [inline]`

Gives the full name of the base, Base::name.

Returns

The full name of the base

6.2.3.7   getVector()   `OpenBabel::OBAtom* PNAB::Base::getVector ( )  [inline]`

Gives the (most likely hydrogen) atom of the base connected to the atom from getLinker() which defines how the base connects, Base::linker[1].

Returns

A pointer to the atom forming the vector connecting to the backbone

6.2.3.8   validate()   `void Base::validate ( )  [private]`

Does some basic sanity checks (such as whether or not the indices of the atom are within the range of the molecule).

6.2.4   Member Data Documentation

6.2.4.1   base   `OpenBabel::OBMol PNAB::Base::base`

The OBMol defining the base.

6.2.4.2   code   `std::string PNAB::Base::code`

Three character code to define base ("Adenine": "ADE")

6.2.4.3 file_path `std::string PNAB::Base::file_path`

Path to a file containing the base.

6.2.4.4 linker `std::array<std::size_t, 2 > PNAB::Base::linker`

Holds indices for atoms forming a vector to connect to backbone {linker, hydrogen}.

6.2.4.5 name `std::string PNAB::Base::name`

Full name of base (i.e. "Adenine" or just "A")

6.2.4.6 pair_name `std::string PNAB::Base::pair_name`

Name of the pair base.

6.2.4.7 vector_atom_deleted `bool PNAB::Base::vector_atom_deleted  [private]`

Whether or not the getVector() atom was deleted.

The documentation for this class was generated from the following files:

- Containers.h
- Containers.cpp

6.3 PNAB::Bases Class Reference

A class that contains a vector of all the defined bases and a funtion to return a base and the complimentary base for all the bases defined.

`#include <Containers.h>`

Public Member Functions

- Bases (std::vector< Base > input_bases)

  Basic constructor for the Bases.
- Bases ()

  Empty constructor.
- PNAB::Base getBaseFromName (std::string name)

  Returns the Base instance given the name of the base.
- std::vector< Base > getBasesFromStrand (std::vector< std::string > strand)

  Returns the vector of the instances of Base given the names of the bases in the strand.
- std::vector< Base > getComplimentBasesFromStrand (std::vector< std::string > strand)

  Returns the complimentary vector of the instances of Base given the names of the bases in the strand.

Private Attributes

- std::vector< Base > bases

    The vector of bases.
- bool all_bases_pair

    Whether all the bases in the strand have complimentary bases.
- std::map< std::string, PNAB::Base > name_base_map

    A map of the names of the bases and the complimentary bases.

6.3.1 Detailed Description

A class that contains a vector of all the defined bases and a funtion to return a base and the complimentary base for all the bases defined.

See also

 Base

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Bases() [1/2] `Bases::Bases (`
            `std::vector< Base > input_bases )`

Basic constructor for the Bases.

The given bases are processed to create a vector of the defined bases and a vector of the complimentary bases. This calls Base to make sure the given bases have Openbabel molecules

Parameters

input_bases	The vector containing the information about the bases needed

6.3.2.2 Bases() [2/2] `PNAB::Bases::Bases ( )` `[inline]`

Empty constructor.

6.3.3 Member Function Documentation

6.3.3.1 getBaseFromName() `PNAB::Base PNAB::Bases::getBaseFromName (`
            `std::string name ) ` `[inline]`

Returns the Base instance given the name of the base.

Parameters

name	name of the base

Returns

the base

6.3.3.2 getBasesFromStrand() `std::vector< Base > Bases::getBasesFromStrand (`
`            std::vector< std::string > strand )`

Returns the vector of the instances of Base given the names of the bases in the strand.

Parameters

strand	vector of base names in the strand

Returns

vector of bases in the strand

See also

Chain::Chain

6.3.3.3 getComplimentBasesFromStrand() `std::vector< Base > Bases::getComplimentBasesFrom↩`
`Strand (`
`            std::vector< std::string > strand )`

Returns the complimentary vector of the instances of Base given the names of the bases in the strand.

Parameters

strand	vector of base names in the strand

Returns

vector of bases in the complimentary strand

See also

Chain::Chain

6.3.4 Member Data Documentation

6.3.4.1 all_bases_pair `bool PNAB::Bases::all_bases_pair [private]`

Whether all the bases in the strand have complimentary bases.

6.3.4.2 bases `std::vector<Base> PNAB::Bases::bases [private]`

The vector of bases.

6.3.4.3 name_base_map `std::map<std::string, PNAB::Base> PNAB::Bases::name_base_map [private]`

A map of the names of the bases and the complimentary bases.

The documentation for this class was generated from the following files:

- Containers.h
- Containers.cpp

6.4 PNAB::BaseUnit Class Reference

Class to hold bases with backbones attached (nucleotides), along with associated necessary information.

`#include <Containers.h>`

Public Member Functions

- BaseUnit (Base b, Backbone backbone, double glycosidic_bond_distance=0.0)
- BaseUnit ()

    Empty constructor.
- const OpenBabel::OBMol getMol ()

    Returns the nucleotide molecule, BaseUnit::unit.
- const std::array< std::size_t, 2 > getBaseIndexRange ()

    Returns the indices for the begining and end of the nucleobase atom indices, BaseUnit::base_index_range.
- const std::array< std::size_t, 2 > getBackboneIndexRange ()

    Returns the indices for the begining and end of the backbone atom indices, BaseUnit::backbone_index_range.
- const std::array< std::size_t, 2 > getBackboneLinkers ()

    Returns the indices for atoms where the backbone connects, BaseUnit::backbone_interconnects.
- std::size_t getBaseConnectIndex ()

    Returns the index of the atom where the nucleobase connects to the backbone, BaseUnit::base_connect_index.
- std::vector< std::vector< unsigned > > getFixedBonds ()

    Returns a vector of the pair of indices for fixed rotatable dihedrals in the backbone, BaseUnit::fixed_bonds.

Private Attributes

- OpenBabel::OBMol unit

  Holds molecule containing base with backbone attached.
- std::array< std::size_t, 2 > base_index_range

  Range of indices of the unit that are a part of the base, [start, stop].
- std::array< std::size_t, 2 > backbone_index_range

  Range of indices of the unit that are a part of the backbone, [start, stop].
- std::size_t base_connect_index

  Atom index where backbone connects to base (the base atom)
- std::array< std::size_t, 2 > backbone_interconnects

  Atom indices defining where backbone connects.
- std::vector< std::vector< unsigned > > fixed_bonds

  Indices of fixed bonds in dihedral search.

6.4.1 Detailed Description

Class to hold bases with backbones attached (nucleotides), along with associated necessary information.

See also

> Base
>
> Backbone
>
> Chain::setupChain
>
> ConformationSearch

6.4.2 Constructor & Destructor Documentation

6.4.2.1 BaseUnit() [1/2] `BaseUnit::BaseUnit (`
            `Base b,`
            `Backbone backbone,`
            `double glycosidic_bond_distance = 0.0 )`

Constructor for the base unit

Parameters

b	Base instance
backbone	Backbone instance
glycosidic_bond_distance	The glycosidic bond distance. If zero (default), set it by using the van der Waals radii

6.4.2.2 BaseUnit() [2/2] `PNAB::BaseUnit::BaseUnit ( )` `[inline]`

Empty constructor.

6.4.3   Member Function Documentation

6.4.3.1   getBackboneIndexRange() `const std::array< std::size_t, 2 > PNAB::BaseUnit::get`↩
`BackboneIndexRange ( )  [inline]`

Returns the indices for the begining and end of the backbone atom indices, BaseUnit::backbone_index_range.

Returns

The indices for the limits of the backbone

6.4.3.2   getBackboneLinkers() `const std::array< std::size_t, 2 > PNAB::BaseUnit::getBackbone`↩
`Linkers ( )  [inline]`

Returns the indices for atoms where the backbone connects, BaseUnit::backbone_interconnects.

Returns

The indices for atoms where the backbone connects

6.4.3.3   getBaseConnectIndex() `std::size_t PNAB::BaseUnit::getBaseConnectIndex ( )  [inline]`

Returns the index of the atom where the nucleobase connects to the backbone, BaseUnit::base_connect_index.

Returns

The nucleobase index connecting to the backbone

6.4.3.4   getBaseIndexRange() `const std::array< std::size_t, 2 > PNAB::BaseUnit::getBaseIndex`↩
`Range ( )  [inline]`

Returns the indices for the begining and end of the nucleobase atom indices, BaseUnit::base_index_range.

Returns

The indices for the limits of the nucleobases

6.4.3.5 getFixedBonds() `std::vector<std::vector<unsigned> > PNAB::BaseUnit::getFixedBonds ( )` `[inline]`

Returns a vector of the pair of indices for fixed rotatable dihedrals in the backbone, BaseUnit::fixed_bonds.

Returns

    The indices of the fixed rotatable dihedrals

6.4.3.6 getMol() `const OpenBabel::OBMol PNAB::BaseUnit::getMol ( )` `[inline]`

Returns the nucleotide molecule, BaseUnit::unit.

Returns

    The nucleotide molecule

6.4.4 Member Data Documentation

6.4.4.1 backbone_index_range `std::array< std::size_t, 2 > PNAB::BaseUnit::backbone_index_↩` `range` `[private]`

Range of indices of the unit that are a part of the backbone, [start, stop].

6.4.4.2 backbone_interconnects `std::array< std::size_t, 2 > PNAB::BaseUnit::backbone_interconnects` `[private]`

Atom indices defining where backbone connects.

6.4.4.3 base_connect_index `std::size_t PNAB::BaseUnit::base_connect_index` `[private]`

Atom index where backbone connects to base (the base atom)

6.4.4.4 base_index_range `std::array< std::size_t, 2 > PNAB::BaseUnit::base_index_range` `[private]`

Range of indices of the unit that are a part of the base, [start, stop].

6.4.4.5 fixed_bonds `std::vector<std::vector<unsigned> > PNAB::BaseUnit::fixed_bonds [private]`

Indices of fixed bonds in dihedral search.

6.4.4.6 unit `OpenBabel::OBMol PNAB::BaseUnit::unit [private]`

Holds molecule containing base with backbone attached.

The documentation for this class was generated from the following files:

- Containers.h
- Containers.cpp

6.5 PNAB::Chain Class Reference

A class for building nucleic acid strands and evaluating their energies.

```
#include <Chain.h>
```

Public Member Functions

- Chain (PNAB::Bases bases, const PNAB::Backbone &backbone, std::vector< std::string > strand, std←
  ::string ff_type, std::array< unsigned, 2 > &range, bool hexad, std::vector< bool > build_strand={true, false,
  false, false, false, false}, std::vector< bool > strand_orientation={true, true, true, true, true, true}, double
  glycosidic_bond_distance=0.0)

  Constructor for the chain class.
- ∼Chain ()

  Destructor for the chain class.
- PNAB::ConformerData generateConformerData (double ∗xyz, PNAB::HelicalParameters &hp, std::vector<
  double > energy_filter)

  Generate structure and energy data for nucleic acid conformers.

Private Member Functions

- void fillConformerEnergyData (double ∗xyz, PNAB::ConformerData &conf_data, std::vector< double >
  energy_filter)

  Computes the energy terms for the candidate system and determines whether it satisfies the energy thresholds.
- void setupChain (std::vector< PNAB::Base > &strand, OpenBabel::OBMol &chain, std::vector< unsigned >
  &new_bond_ids, std::vector< unsigned > &deleted_atoms_ids, std::vector< unsigned > &num_base_unit←
  _atoms, std::vector< unsigned > &bb_start_index, std::vector< double ∗ > &base_coords_vec, std::vector<
  std::vector< unsigned >> &fixed_bonds_vec, const Backbone &backbone, unsigned chain_index)

  Creates the molecule for each strand in the system.
- void setupFFConstraints (OpenBabel::OBMol &chain, std::vector< unsigned > &new_bond_ids, std::vector<
  std::vector< unsigned >> &fixed_bonds_vec, unsigned offset=0)

  Determines the terms that should be ignored during the computation of the bond, angle, and torsional energies.
- void setCoordsForChain (double ∗xyz, double ∗conf, PNAB::HelicalParameters &hp, std::vector< unsigned
  > &num_bu_atoms, std::vector< unsigned > &bb_start_index, std::vector< double ∗ > &base_coords_vec,
  std::vector< unsigned > &deleted_atoms_ids, unsigned chain_index)

  Set the coordinates for each strand in the system.
- void orderResidues (OpenBabel::OBMol ∗molecule)

  Orders the residues in the molecules correctly.

Private Attributes

- std::vector< OpenBabel::OBMol > v_chain_ = std::vector<OpenBabel::OBMol>(6)

  A vector of OpenBabel::OBMol containing the molecules for each strand in the system.
- OpenBabel::OBMol combined_chain_

  An OpenBabel::OBMol molecule containing the structure of the whole system.
- std::vector< std::vector< unsigned > > v_new_bond_ids_ = std::vector<std::vector<unsigned>>(6)

  a vector containing a vector of the IDs of the atoms forming new bonds between the nucleotides in each strand
- std::vector< std::vector< unsigned > > v_deleted_atoms_ids_ = std::vector<std::vector<unsigned>>(6)

  A vector containing a vector of the IDs of the atoms deleted in each strand because of the formation of new bonds.
- std::vector< std::vector< unsigned > > v_num_bu_A_mol_atoms_ = std::vector<std::vector<unsigned>>(6)

  A vector containing a vector of the number of atoms in each BaseUnit for each strand.
- std::vector< std::vector< std::vector< unsigned > > > v_fixed_bonds = std::vector<std::vector<std↵::vector<unsigned>>>(6)

  A vector containing the indices of fixed rotatable bonds for each strad.
- std::vector< std::vector< double * > > v_base_coords_vec_ = std::vector<std::vector<double∗>>(6)

  A vector containing a vector the coordinates of each nucleotide in each strand.
- unsigned chain_length_

  The number of nucleotides in the strand.
- unsigned n_chains_

  The number of strands in the system.
- bool isKCAL_

  Whether the energy computed by openbabel is in kcal/mol.
- bool hexad_

  Whether we are building a hexad, RuntimeParameters::is_hexad.
- std::vector< bool > strand_orientation_

  A vector containing the orientation of each strand in the hexad, RuntimeParameters::strand_orientation.
- double glycosidic_bond_distance_

  The distance of the glycosidic bond, RuntimeParameters::glycosidic_bond_distance.
- OpenBabel::OBForceField ∗ pFF_

  The openbabel force field. Used to compute the energy of the system.
- std::array< unsigned, 2 > monomer_bb_index_range_

  Backbone index range for the first nucleotide.
- std::vector< std::vector< unsigned > > v_bb_start_index_ = std::vector<std::vector<unsigned>>(6)

  A vector containing a vector of the starting indices of the backbone atoms in the BaseUnit for each strand.
- std::string ff_type_

  The force field type (e.g. "GAFF"), RuntimeParameters::ff_type.
- std::vector< std::vector< unsigned int > > all_angles_

  A vector of the vector of atom indices forming all the angles for which we need to compute the energy.
- std::vector< std::vector< unsigned int > > all_torsions_

  A vector of the vector of atom indices forming all the torsions for which we need to compute the energy.
- std::vector< bool > is_fixed_bond

  A vector containing whether the torsional energy term is for a fixed rotatable bond or not.
- std::vector< bool > build_strand_

  A vector containing whether a given strand should be built.
- OpenBabel::OBFFConstraints constraintsBond_

  Setting all atoms not forming the new bond between the first two nucleotides to be ignored during bond energy computation.
- OpenBabel::OBFFConstraints constraintsAng_

  An empty constraint object for angles; Energy groups are used for the angle terms.
- OpenBabel::OBFFConstraints constraintsTor_

  An empty constraint object for torsions; Energy groups are used for the torsion terms.
- OpenBabel::OBFFConstraints constraintsTot_

  An empty constraint object for van der Waals and total energy terms. No ignored atoms in these computations.

6.5.1   Detailed Description

A class for building nucleic acid strands and evaluating their energies.

The class creates the strands by connecting the nucleotides created in the BaseUnit class. It forms bonds between the nucleotides for the given sequence of the nucleobases, and creates duplex or hexad systems if requested. The class also contains functions for computing the energy terms for the system and checking whether the candidates are accepted. Several functions in the class identifies the proper energy terms to be computed for the system (e.g. bond and angle energies).

See also

> ConformationSearch
>
> HelicalParameters
>
> RuntimeParameters
>
> BaseUnit
>
> Bases

6.5.2   Constructor & Destructor Documentation

6.5.2.1   Chain()   `Chain::Chain (`
> `PNAB::Bases bases,`
> `const PNAB::Backbone & backbone,`
> `std::vector< std::string > strand,`
> `std::string ff_type,`
> `std::array< unsigned, 2 > & range,`
> `bool hexad,`
> `std::vector< bool > build_strand = {true, false, false, false, false, false},`
> `std::vector< bool > strand_orientation = {true, true, true, true, true, true},`
> `double glycosidic_bond_distance = 0.0 )`

Constructor for the chain class.

The constructor performs several tasks. It creates an OpenBabel::ForceField object and check that it works as expected. It identifies whether we are working with the canonical nucleobases or with the hexad. For double-stranded and hexad systems, it checks whether the user provided correct comlimentary bases. After all these checks, it proceeds to setup the requested molecules. The constructor does not put correct coordinates for the molecules, but creates the topology of each strand in the system. The constructor also identifies the atoms that should be ignored when computing the separate energies (e.g. bond energy) for the backbone candidates.

Parameters

bases	A vector of the all the defined bases,
backbone	The backbone
strand	A vector of the names of all bases in the strand, RuntimeParameters::strand
ff_type	The force field type, RuntimeParameters::ff_type
range	Backbone index range for the first nucleotide
hexad	Defines whether the 60 degrees rotation for hexads is performed, RuntimeParameters::is_hexad
build_strand	Defines whether to build a given strand, RuntimeParameters::build_strand
strand_orientation	The orientation of each strand in the hexad, RuntimeParameters::strand_orientation
glycosidic_bond_distance	The distance of the glycosidic bond, RuntimeParameters::glycosidic_bond_distance

See also

> setupChain
>
> setupFFConstraints

6.5.2.2 ∼Chain() `PNAB::Chain::∼Chain ( )` `[inline]`

Destructor for the chain class.

6.5.3 Member Function Documentation

6.5.3.1 fillConformerEnergyData() `void Chain::fillConformerEnergyData (`
> `double * xyz,`
> `PNAB::ConformerData & conf_data,`
> `std::vector< double > energy_filter ) [private]`

Computes the energy terms for the candidate system and determines whether it satisfies the energy thresholds.

This function uses openbabel to compute the energy for the candidate systems. See the description in setupF←
FConstraints for the bond, angle, and torsional energies terms. The van der Waals and total energy terms of the
systems are computed without any constraints. The energy terms are computed sequentially as follows: bond,
angle, torsion, van der Waals, and total energies. If any one of the energy terms does not satisfy the energy
thresholds defined in RuntimeParameters::energy_filter, then the candidate is rejected, and we do not proceed to
compute the additional energy terms.

Parameters

xyz	The coordinates of the whole system
conf_data	An object that will hold the values of the energy terms, the coordinates, and whether it is accepted.
energy_filter	The energy thresholds for the system, RuntimeParameters::energy_filter

See also

> generateConformerData

6.5.3.2 generateConformerData() `ConformerData Chain::generateConformerData (`
> `double * xyz,`
> `PNAB::HelicalParameters & hp,`
> `std::vector< double > energy_filter )`

Generate structure and energy data for nucleic acid conformers.

After the conformation search finds a candidate that satisfies the distance threshold, this function is called with the
coordinates of the backbone candidate to build the full system and computes its energy properties.

This function calls Chain::setCoordsForChain and Chain::fillConformerEnergyData to setup the coordinates and
compute the energies, respectively.

Parameters

xyz	The coordinates of the backbone candidate that satisfied the distance threshold
hp	An instance of HelicalParameters that contains the helical structure data and functions
energy_filter	The energy thresholds for the system, RuntimeParameters::energy_filter

Returns

The conformer energy data, ConformerData, containing whether the candidate is accepted

See also

setCoordsForChain

fillConformerEnergyData

ConformationSearch::RandomSearch

ConformationSearch::MonteCarloSearch

ConformationSearch::SystematicSearch

ConformationSearch::GeneticAlgorithmSearch

6.5.3.3 orderResidues() `void Chain::orderResidues (`
`          OpenBabel::OBMol * molecule ) [private]`

Orders the residues in the molecules correctly.

This function reverses the order of residues in inverted strands.

Parameters

molecule	The OBMol object of the system which will get the correct order

See also

generateConformerData

PNAB::ConformerData

6.5.3.4 setCoordsForChain() `void Chain::setCoordsForChain (`
`          double * xyz,`
`          double * conf,`
`          PNAB::HelicalParameters & hp,`
`          std::vector< unsigned > & num_bu_atoms,`
`          std::vector< unsigned > & bb_start_index,`
`          std::vector< double * > & base_coords_vec,`
`          std::vector< unsigned > & deleted_atoms_ids,`
`          unsigned chain_index ) [private]`

Set the coordinates for each strand in the system.

This function set the coordinates for each nucleotide in the strand. For DNA- and RNA- like systems, this function can generate anti-parallel strands. For hexad systems, it can generate any combination of the parallel and anti-parallel strands.

Parameters

xyz	An array that will get the coordinates of the strands
conf	An array containing the coordinates of one backbone determined by the search algorithms
hp	An instance of HelicalParameters which has the helical parameters and functions to generate the coordinates
num_bu_atoms	The number of atoms in each BaseUnit
bb_start_index	A vector containing the starting indices of the backbone atoms in the BaseUnit
base_coords_vec	A vector containing the coordinates of each one of the nucleotides
deleted_atoms_ids	A vector containing the indices of the atoms deleted because of the bonding between the nucleotides
chain_index	The index of the strand in the system

See also

> setupChain
>
> generateConformerData
>
> HelicalParameters
>
> ConformationSearch

6.5.3.5 setupChain() `void Chain::setupChain (`
`        std::vector< PNAB::Base > & strand,`
`        OpenBabel::OBMol & chain,`
`        std::vector< unsigned > & new_bond_ids,`
`        std::vector< unsigned > & deleted_atoms_ids,`
`        std::vector< unsigned > & num_base_unit_atoms,`
`        std::vector< unsigned > & bb_start_index,`
`        std::vector< double * > & base_coords_vec,`
`        std::vector< std::vector< unsigned >> & fixed_bonds_vec,`
`        const Backbone & backbone,`
`        unsigned chain_index )  [private]`

Creates the molecule for each strand in the system.

This function creates the topology (makes and removes bonds) for each strand in the system. It also determines the indices for all the fixed rotatable bonds in the system.

Parameters

strand	The list of bases in the strand
chain	An empty openbabel molecule to be populated with the topology of the strand
new_bond_ids	An empty vector to be populated with the indices of the atoms forming bonds between the nucleotides

Parameters

deleted_atoms_ids	An empty vector to be populated with the indices of the atoms deleted because of the bonding between the nucleotides
num_base_unit_atoms	An empty vector to be populated with the number of atoms in each BaseUnit
bb_start_index	An empty vector to be populated with the starting index of the backbone atoms in the BaseUnit
base_coords_vec	An empty vector to be populated with the coordinates of each one of the nucleotides
fixed_bonds_vec	An empty vector to be populated with the indices of fixed bonds in each one of the nucleotides
backbone	An instance of the Backbone class
chain_index	The index of the chain

See also

> Chain::Chain

6.5.3.6   setupFFConstraints() `void Chain::setupFFConstraints (`

```
OpenBabel::OBMol & chain,
std::vector< unsigned > & new_bond_ids,
std::vector< std::vector< unsigned >> & fixed_bonds_vec,
unsigned offset = 0 )  [private]
```

Determines the terms that should be ignored during the computation of the bond, angle, and torsional energies.

For the bond energy, we consider only the one new bond that is formed between the first and second nucleobases in the system. This bond energy is the same for all new bonds, so we do not need to compute it for all of them.

For the angle energy, we consider all the new angles that are formed between the first and second nucleobases in the system. This angle energy is the same for all new bonds, so we do not need to compute it for all of them.

For the torsional energy, we consider all the rotatable bonds in all the nucleobases. There will be some redundant calculations, as some torsional angles will be the same. This code also identifies whether a rotatable torsional angle is fixed or not. For the identified torsional angles, we do not include any torsional angle that runs between the residues, as these are not rotated in the search procedure.

Parameters

chain	The openbabel molecule that contain the topology of the strand
new_bond_ids	A vector containing the indices of the atoms forming bonds between the nucleotides
fixed_bonds_vec	A vector containing the indices of fixed bonds in each one of the nucleotides
offset	The number of atoms that have been processed previously. Used to set correct indices for all the strands.

See also

> Chain::Chain
>
> setupChain
>
> fillConformerEnergyData

6.5.4 Member Data Documentation

6.5.4.1 all_angles_ `std::vector<std::vector<unsigned int> > PNAB::Chain::all_angles_ [private]`

A vector of the vector of atom indices forming all the angles for which we need to compute the energy.

6.5.4.2 all_torsions_ `std::vector<std::vector<unsigned int> > PNAB::Chain::all_torsions_←` `[private]`

A vector of the vector of atom indices forming all the torsions for which we need to compute the energy.

6.5.4.3 build_strand_ `std::vector<bool> PNAB::Chain::build_strand_ [private]`

A vector containing whether a given strand should be built.

6.5.4.4 chain_length_ `unsigned PNAB::Chain::chain_length_ [private]`

The number of nucleotides in the strand.

6.5.4.5 combined_chain_ `OpenBabel::OBMol PNAB::Chain::combined_chain_ [private]`

An OpenBabel::OBMol molecule containing the structure of the whole system.

6.5.4.6 constraintsAng_ `OpenBabel::OBFFConstraints PNAB::Chain::constraintsAng_ [private]`

An empty constraint object for angles; Energy groups are used for the angle terms.

6.5.4.7 constraintsBond_ `OpenBabel::OBFFConstraints PNAB::Chain::constraintsBond_ [private]`

Setting all atoms not forming the new bond between the first two nucleotides to be ignored during bond energy computation.

6.5.4.8 constraintsTor_ `OpenBabel::OBFFConstraints PNAB::Chain::constraintsTor_ [private]`

An empty constraint object for torsions; Energy groups are used for the torsion terms.

6.5.4.9 constraintsTot_ `OpenBabel::OBFFConstraints PNAB::Chain::constraintsTot_ [private]`

An empty constraint object for van der Waals and total energy terms. No ignored atoms in these computations.

6.5.4.10 ff_type_ `std::string PNAB::Chain::ff_type_ [private]`

The force field type (e.g. "GAFF"), RuntimeParameters::ff_type.

6.5.4.11 glycosidic_bond_distance_ `double PNAB::Chain::glycosidic_bond_distance_ [private]`

The distance of the glycosidic bond, RuntimeParameters::glycosidic_bond_distance.

6.5.4.12 hexad_ `bool PNAB::Chain::hexad_ [private]`

Whether we are building a hexad, RuntimeParameters::is_hexad.

6.5.4.13 is_fixed_bond `std::vector<bool> PNAB::Chain::is_fixed_bond [private]`

A vector containing whether the torsional energy term is for a fixed rotatable bond or not.

6.5.4.14 isKCAL_ `bool PNAB::Chain::isKCAL_ [private]`

Whether the energy computed by openbabel is in kcal/mol.

6.5.4.15 monomer_bb_index_range_ `std::array<unsigned, 2> PNAB::Chain::monomer_bb_index_↩ range_ [private]`

Backbone index range for the first nucleotide.

6.5.4.16 n_chains_ `unsigned PNAB::Chain::n_chains_` `[private]`

The number of strands in the system.

6.5.4.17 pFF_ `OpenBabel::OBForceField* PNAB::Chain::pFF_` `[private]`

The openbabel force field. Used to compute the energy of the system.

6.5.4.18 strand_orientation_ `std::vector<bool> PNAB::Chain::strand_orientation_` `[private]`

A vector containing the orientation of each strand in the hexad, RuntimeParameters::strand_orientation.

6.5.4.19 v_base_coords_vec_ `std::vector<std::vector<double*> > PNAB::Chain::v_base_coords_←↵
vec_ = std::vector<std::vector<double*>>(6)` `[private]`

A vector containing a vector the coordinates of each nucleotide in each strand.

6.5.4.20 v_bb_start_index_ `std::vector<std::vector<unsigned> > PNAB::Chain::v_bb_start_←↵
index_ = std::vector<std::vector<unsigned>>(6)` `[private]`

A vector containing a vector of the starting indices of the backbone atoms in the BaseUnit for each strand.

6.5.4.21 v_chain_ `std::vector<OpenBabel::OBMol> PNAB::Chain::v_chain_ = std::vector<Open←↵
Babel::OBMol>(6)` `[private]`

A vector of OpenBabel::OBMol containing the molecules for each strand in the system.

6.5.4.22 v_deleted_atoms_ids_ `std::vector<std::vector<unsigned> > PNAB::Chain::v_deleted_←↵
atoms_ids_ = std::vector<std::vector<unsigned>>(6)` `[private]`

A vector containing a vector of the IDs of the atoms deleted in each strand because of the formation of new bonds.

6.5.4.23 v_fixed_bonds `std::vector<std::vector<std::vector<unsigned> > > PNAB::Chain::v_←↵
fixed_bonds = std::vector<std::vector<std::vector<unsigned>>>(6)` `[private]`

A vector containing the indices of fixed rotatable bonds for each strad.

6.5.4.24 v_new_bond_ids_ `std::vector<std::vector<unsigned> > PNAB::Chain::v_new_bond_ids_ = std::vector<std::vector<unsigned>>(6)` `[private]`

a vector containing a vector of the IDs of the atoms forming new bonds between the nucleotides in each strand

6.5.4.25 v_num_bu_A_mol_atoms_ `std::vector<std::vector<unsigned> > PNAB::Chain::v_num_bu↩ _A_mol_atoms_ = std::vector<std::vector<unsigned>>(6)` `[private]`

A vector containing a vector of the number of atoms in each BaseUnit for each strand.

The documentation for this class was generated from the following files:

- Chain.h
- Chain.cpp

6.6 PNAB::ConformationSearch Class Reference

A rotor search function used to find acceptable conformations of arbitrary backbone and helical parameter combinations. The main class of the proto-Nucleic Acid Builder.

```
#include <ConformationSearch.h>
```

Collaboration diagram for PNAB::ConformationSearch:
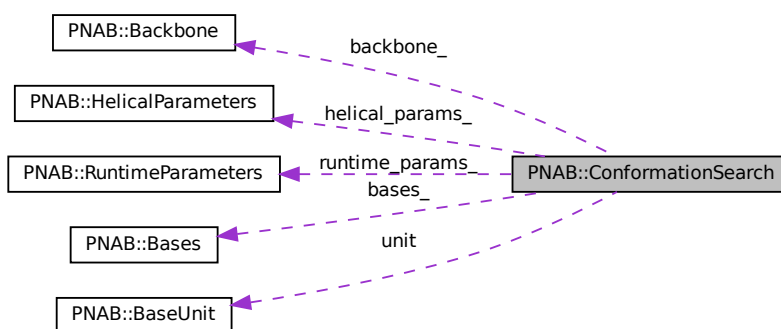
Public Member Functions

- ConformationSearch (PNAB::RuntimeParameters &runtime_params, PNAB::Backbone &backbone, PNAB::HelicalParameters &helical_params, PNAB::Bases bases, std::string prefix="test", bool verbose=true)

  Constructor for the conformation search class.
- std::string run ()

  A function to call the appropriate search algorithm using the provided RuntimeParameters::search_algorithm.

Private Member Functions

- void SystematicSearch ()

  Given a step size, the algorithm exhaustively searches over all the rotatable dihedral angles in the backbone.
- void RandomSearch (bool weighted)

  The algorithm randomly changes all the dihedral angles in the backbone and evaluates whether they are acceptable.
- void MonteCarloSearch (bool weighted)

  The algorithm utilizes the Metropolis Monte Carlo scheme to improve the choice of the dihedral angle.
- void GeneticAlgorithmSearch ()

  This algorithm utilizes the genetic algorihtm procedure to improve the choice of the dihedral angle.
- std::vector< std::piecewise_linear_distribution< double > > WeightedDistributions ()

  Produces weighted distributions for each rotatable dihedral angle in the backbone.
- double measureDistance (double *coords, unsigned head, unsigned tail)

  Compute the distance between the head backbone atom and the tail backbone atom of the next nucleotide.
- void reportData (PNAB::ConformerData &conf_data)

  A function to report the data on the accepted candidates.
- void printProgress (std::size_t search_index, std::size_t search_size)

  Prints the percentage of search completed and the best accepted candidate.

Private Attributes

- bool verbose_

  Whether to print progress report to screen.
- PNAB::RuntimeParameters runtime_params_

  The runtime parameters instance, RuntimeParameters.
- std::array< unsigned, 2 > backbone_range_

  The Backbone index range for the first nucleotide.
- PNAB::Backbone backbone_

  The backbone molecule.
- PNAB::HelicalParameters helical_params_

  the helical parameters
- PNAB::Bases bases_

  the list of the defined bases
- std::mt19937_64 rng_

  A random number generator.
- int number_of_candidates = 0

  The number of accepted candiates.
- OpenBabel::matrix3x3 step_rot_

  The step rotation matrix, HelicalParameters::getStepRotationMatrix.
- OpenBabel::matrix3x3 glbl_rot_

  The global rotation matrix, HelicalParameters::getGlobalRotationMatrix.
- OpenBabel::vector3 step_translate_

  The step translation vector, HelicalParameters::getStepTranslationVec.
- OpenBabel::vector3 glbl_translate_

  The global translation vector, HelicalParameters::getGlobalTranslationVec.
- OpenBabel::OBConversion conv_

  An openbabel conversion object used to write accepted candidates.
- unsigned monomer_num_coords_

  *3*The number of atoms in the first nucleotide*
- std::string prefix_

A string to prepend the name of the accepted backbone candidates.

- BaseUnit unit

    The nucleotide unit for the first nucleotide in the strand. Used for searching for conformations.

- OpenBabel::OBMol bu_a_mol

    The molecule of the first nucleotide.

- unsigned head

    The first terminal atom in the backbone.

- unsigned tail

    The second terminal atom in the backbone.

- std::string output_string

    A string in CSV format containing the properties of the accepted candidates.

- double ∗ coords

    The coordinates of the first nucleotide.

- OpenBabel::OBRotorList rl

    The list of the all rotatable dihedral angles in the backbone.

- std::vector< OpenBabel::OBRotor ∗ > rotor_vector

    A vector of dihedral angles to be rotated in the search. Excludes fixed angles.

6.6.1   Detailed Description

A rotor search function used to find acceptable conformations of arbitrary backbone and helical parameter combinations. The main class of the proto-Nucleic Acid Builder.

The general algorithm follows the following steps:

1. Define the input parameters:

    - The three-dimensional structure of the backbone, the atoms that form the connection to the base, and the two atoms (head and tail) that link the backbone with two adjacent backbones.
    - The helical configuration of the nucleobases.
    - The runtime parameters:
        - The specific search algorithm (see below).
        - The distance and energy thresholds (see below).
        - The sequence of the nucleobases and whether the system is single-stranded, a duplex, or a hexad.

2. Connect the backbone with the first nucleobase in the sequence.

3. Fix all the bond distances, bond angles, and the rigid dihedral angles in the backbone. Fix all the atoms of the nucleobase.

4. Change the dihdedal angles in the backbone using one of the search algorithm.

5. Translate and rotate the tail atom of the backbone using the given helical parameters.

6. Measure the distance between the head and the tail atoms of the backbone.

7. If the distance is greater than the distance threshold, go to 4. Else, proceed.

8. Arrange the nucleobases using the given helical parameters. Replicate the backbone structure across all the nucleotides.

9. Evaluate various energy terms for the system.

10. If the energy terms are less than the energy thresholds, save the structure. Else, reject the structure.

11. Repeat steps 4-10 or quit.

The various search algorithm differ in the implementation of step 4. Four classes of search algorithms are implemented here, namely:

- Systematic Search

- Random Search

- Monte Carlo Search

- Genetic Algorithm Search

Details on the specific implementation of each algorithm are discussed below.

See also

> SystematicSearch
>
> RandomSearch
>
> MonteCarloSearch
>
> GeneticAlgorithmSearch
>
> HelicalParameters
>
> RuntimeParameters
>
> Backbone
>
> BaseUnit
>
> Chain

6.6.2 Constructor & Destructor Documentation

6.6.2.1 ConformationSearch() `ConformationSearch::ConformationSearch (`
```
        PNAB::RuntimeParameters & runtime_params,
        PNAB::Backbone & backbone,
        PNAB::HelicalParameters & helical_params,
        PNAB::Bases bases,
        std::string prefix = "test",
        bool verbose = true )
```

Constructor for the conformation search class.

The constructor takes the input parameters and constructs a single nucleotide. Then, it determines the rotatable dihedral angles in the backbone.

Parameters

runtime_params	Series of parameters that control how the search algorithm runs
backbone	The backbone molecule over which the algorithm searches
helical_params	The geometric parameters constraining the possible conformations of the backbone
bases	List of defined bases that serves of as the library used for building the strand
prefix	A string to prepend the name of the accepted backbone candidates
verbose	Whether to print progress report to screen

6.6.3 Member Function Documentation

6.6.3.1 GeneticAlgorithmSearch() `void ConformationSearch::GeneticAlgorithmSearch ( ) [private]`

This algorithm utilizes the genetic algorihtm procedure to improve the choice of the dihedral angle.

In the genetic algorithm search, a population of rotamers is initialized with random dihedral angles. At each iteration (or generation), certain rotamers (parents) are chosen for reproduction based on their fitness. The offsprings are generated by allowing the two parents to exchange (crossover) one dihedral angle. Then, a random mutation (or a new value for the dihedral angle) may be introduced for one dihedral angle. The new population consisting of the offsprings then follow the same procedure. The algorithm finishes after the specified number of generations. The length of the search is proportional to the number of generations multiplied by the size of the population.

The fitness of the individuals is computed as the inverse of the distance between the head atom of the backbone and the tail atom of the adjacent backbone. The individuals in the population are ranked by their fitness score, and the probability of choosing individuals for mating is proportional to their ranking.

The exchange of the dihedral angles between the parents and the introduction of new dihedral angles is determined by the crossover and mutation probabilities.

The algorithm significantly accelerates the finding of dihedral angles that satisfy the distance threshold. However, the distance threshold only indicates that the candidate backbone configuration can form a periodic structure. It does not indicates whether the backbone configuration is low or high in energy. Therefore, introducing mutations with higher probability is useful in preventing the formation of a homogeneous population that has low distances but high energies.

The probability of choosing a random angle between 0 and 360° is uniform.

See also

RuntimeParameters::num_steps

RuntimeParameters::population_size

RuntimeParameters::crossover_rate

RuntimeParameters::mutation_rate

measureDistance

Chain::generateConformerData

6.6.3.2 measureDistance() `double ConformationSearch::measureDistance (`
`        double * coords,`
`        unsigned head,`
`        unsigned tail ) [private]`

Compute the distance between the head backbone atom and the tail backbone atom of the next nucleotide.

This function applies the global translation and rotation to both the head and the tail atoms of the first nucleotide. Then, it applies the step translation and rotation to the tail atom to get its coordinates in the second nucleotide. Then, it computes the distance between the two atoms. This function is called by all the search algorithms.

Parameters

coords	The coordinates of the first nucleotide
head	The index of the first terminal atom in the backbone of the first nucleotide
tail	The index of the second terminal atom the backbone of the first nucleotide

Returns

The distance between the two atoms in Angstroms

See also

RuntimeParameters::max_distance

ConformationSearch

6.6.3.3 MonteCarloSearch() `void ConformationSearch::MonteCarloSearch (`
`bool weighted ) [private]`

The algorithm utilizes the Metropolis Monte Carlo scheme to improve the choice of the dihedral angle.

At every iteration, this algorithm randomly chooses two dihedrals and set them to random values. If the new configuration decreases the distance between the head atom of the backbone and the tail atom of the adjacent backbone or if the new distance is less than 1 Angstrom, this step is accepted. If the new configuration does not decrease the distance, the step is provisionally accpeted if a random number between 0 and 1 is less than $exp(\frac{E}{k_B T})$, where E is a penalty term based on the distance, and $k_B T$ is the Boltzmann constant multiplied by the temperature. If the step is provisionally accepted, the system is constructed and the energy terms of the system are computed. If the energy terms are higher than the thresholds, the step is finally rejected. The algorithm finishes after the specified number of iterations.

The penalty term E is given by $E = -k(\text{currrent distance} - \text{previous distance})^2$. k is a bond stretching constant fixed at 300 kcal/mol/Angstrom ^{2}. The temperature controls how agressively the algorithm should provisionally accept or reject steps. An infinite temperature will reduce the algorithm to a random search with two dihedral angles changed at every step.

The algorithm significantly accelerates the finding of dihedral angles that satisfy the distance threshold. However, the distance threshold only indicates that the candidate backbone configuration can form a periodic structure. It does not indicates whether the backbone configuration is low or high in energy. Therefore, the ultimate criteria for accepting or rejecting a step, if the distance is not decreased in the step, is whether it satisfies the energy thresholds.

The probability of choosing a random angle between 0 and 360° can be uniform or can be weighted. See the description in WeightedDistributions for an explanation on the weighting scheme for the dihedral angles.

Parameters

weighted	Whether to use a weighted distribution for the dihedral angles

See also

RuntimeParameters::num_steps

RuntimeParameters::weighting_temperature

6.6.3.4   printProgress() `void ConformationSearch::printProgress (`
            `std::size_t` *search_index,*
            `std::size_t` *search_size* `)   [private]`

Prints the percentage of search completed and the best accepted candidate.

Prints to the standard output the percentage of the search completed, the name of the PDB file containing the best accepted candidte, its distance and energy, and the number of accepted candidates.

Parameters

search_index	The step number in the search
search_size	The total number of steps

6.6.3.5   RandomSearch() `void ConformationSearch::RandomSearch (`
            `bool` *weighted* `)   [private]`

The algorithm randomly changes all the dihedral angles in the backbone and evaluates whether they are acceptable.

At every iteration, this algorithm sets all the dihedral angles in the backbone to random values. This algorithm is purely random and each iteration is completely independent from the previous iteration. This algorithm can sample the dihedral angle space much faster than the Systemtic Search algorithm. Therefore, it may find an acceptable candidate faster. However, it is not deterministic. As each iteration is completely independent, this algorithm does not become trapped in unfavorable configurations. Nevertheless, the search does not improve with time. The algorithm finishes after the specified number of iterations.

The probability of choosing a random angle between 0 and 360° can be uniform or can be weighted. See the description in WeightedDistributions for an explanation on the weighting scheme for the dihedral angles.

Parameters

weighted	Whether to use a weighted distribution for the dihedral angles

See also

6.6.3.6 reportData() `void ConformationSearch::reportData (`
`            PNAB::ConformerData & conf_data ) [private]`

A function to report the data on the accepted candidates.

This function saves the structure of each accepted candidate in PDB format. It also reports the properties of the candidates, ordered by their total energies, in a string with the CSV format. This is called by all search algorithms when a candidate is accepted. The ConformationSearch::output_string variable is updated.

Parameters

conf_data	The properties of the accepted candiate

6.6.3.7 run() `std::string ConformationSearch::run ( )`

A function to call the appropriate search algorithm using the provided RuntimeParameters::search_algorithm.

Returns

A string in CSV format containing the properties of the accepted candidates. The structure of the accepted candidates are saved as PDB files.

See also

SystematicSearch

RandomSearch

MonteCarloSearch

GeneticAlgorithmSearch

6.6.3.8 SystematicSearch() `void ConformationSearch::SystematicSearch ( ) [private]`

Given a step size, the algorithm exhaustively searches over all the rotatable dihedral angles in the backbone.

The systematic search algorithm works by exhaustively rotating all the dihedral angles between 0 and 360° using the given step size. This is the simplest search algorithm. However, it is the most time-consuming as the time grows exponentially with the number of rotatable bonds. The number of steps required for the search is $(\frac{360.0}{\text{dihedral step}})^{\text{number of rotatable dihedrals}}$. This algorithm is deterministic and reproducible. It is guranteed to find an acceptable candidate, if any exists, to within the given resolution. This algorithm can be used to validate the results of the other algorithms and to determine whether the other algorithms found all the families of the acceptable candidates.

See also

RuntimeParameters::dihedral_step

measureDistance

Chain::generateConformerData

6.6.3.9   WeightedDistributions() `std::vector< std::piecewise_linear_distribution< double > > ConformationSearch::WeightedDistributions ( )  [private]`

Produces weighted distributions for each rotatable dihedral angle in the backbone.

Instead of using a uniform distrbution for the dihedral angles between 0 and 360°, this function generates weighted distributions for each rotatable dihedral angle in the backbone. The energy of each dihedral angle in isolation is computed every 5°. Then, the probability for a given angle is computed as the $\frac{exp(\frac{E_i}{k_B T})}{\sum_i exp(\frac{E_i}{k_B T})}$, where E_i is the dihedral energy at angle i. The probability in each interval is the linear interpolation of the probabilities at the limits of the interval. The temperature controls how agressively the algorithm should weight the dihedral angles. An infinite temperature will reduce the probabilities to uniformly random distributions.

This weighting scheme is useful in increasing the sampling of the dihedral angles where the energy of the dihedral angle is low. However, the energy of the dihedral angle in isolation does not necessarily mean the nucleotide energy is going to be low. This weighting scheme may improve the search if the acceptable candidates adopt structures where the dihedral angles are not strained. However, it may worsen the search if the unstrained dihedral angles do not produce low-energy candidates.

Returns

A vector of the weighted distributions for each rotatable dihedral angle

See also

RuntimeParameters::weighting_temperature

RandomSearch

MonteCarloSearch

6.6.4   Member Data Documentation

6.6.4.1   backbone_ `PNAB::Backbone PNAB::ConformationSearch::backbone_  [private]`

The backbone molecule.

6.6.4.2   backbone_range_ `std::array<unsigned, 2> PNAB::ConformationSearch::backbone_range_↩ [private]`

The Backbone index range for the first nucleotide.

6.6.4.3   bases_ `PNAB::Bases PNAB::ConformationSearch::bases_  [private]`

the list of the defined bases

6.6.4.4 bu_a_mol `OpenBabel::OBMol PNAB::ConformationSearch::bu_a_mol` `[private]`

The molecule of the first nucleotide.

6.6.4.5 conv_ `OpenBabel::OBConversion PNAB::ConformationSearch::conv_` `[private]`

An openbabel conversion object used to write accepted candidates.

6.6.4.6 coords `double* PNAB::ConformationSearch::coords` `[private]`

The coordinates of the first nucleotide.

6.6.4.7 glbl_rot_ `OpenBabel::matrix3x3 PNAB::ConformationSearch::glbl_rot_` `[private]`

The global rotation matrix, HelicalParameters::getGlobalRotationMatrix.

6.6.4.8 glbl_translate_ `OpenBabel::vector3 PNAB::ConformationSearch::glbl_translate_` `[private]`

The global translation vector, HelicalParameters::getGlobalTranslationVec.

6.6.4.9 head `unsigned PNAB::ConformationSearch::head` `[private]`

The first terminal atom in the backbone.

6.6.4.10 helical_params_ [`PNAB::HelicalParameters`](PNAB::HelicalParameters) `PNAB::ConformationSearch::helical_params_↩` `[private]`

the helical parameters

6.6.4.11 monomer_num_coords_ `unsigned PNAB::ConformationSearch::monomer_num_coords_` `[private]`

3∗The number of atoms in the first nucleotide

6.6.4.12 number_of_candidates `int PNAB::ConformationSearch::number_of_candidates = 0` `[private]`

The number of accepted candiates.

6.6.4.13 output_string `std::string PNAB::ConformationSearch::output_string` `[private]`

A string in CSV format containing the properties of the accepted candidates.

6.6.4.14 prefix_ `std::string PNAB::ConformationSearch::prefix_` `[private]`

A string to prepend the name of the accepted backbone candidates.

6.6.4.15 rl `OpenBabel::OBRotorList PNAB::ConformationSearch::rl` `[private]`

The list of the all rotatable dihedral angles in the backbone.

6.6.4.16 rng_ `std::mt19937_64 PNAB::ConformationSearch::rng_` `[private]`

A random number generator.

6.6.4.17 rotor_vector `std::vector<OpenBabel::OBRotor*> PNAB::ConformationSearch::rotor_vector` `[private]`

A vector of dihedral angles to be rotated in the search. Excludes fixed angles.

6.6.4.18 runtime_params_ [PNAB::RuntimeParameters](#) `PNAB::ConformationSearch::runtime_params_↩` `[private]`

The runtime parameters instance, [RuntimeParameters](#).

6.6.4.19 step_rot_ `OpenBabel::matrix3x3 PNAB::ConformationSearch::step_rot_` `[private]`

The step rotation matrix, [HelicalParameters::getStepRotationMatrix](#).

6.6.4.20   step_translate_   `OpenBabel::vector3 PNAB::ConformationSearch::step_translate_ [private]`

The step translation vector, HelicalParameters::getStepTranslationVec.

6.6.4.21   tail   `unsigned PNAB::ConformationSearch::tail [private]`

The second terminal atom in the backbone.

6.6.4.22   unit   `BaseUnit PNAB::ConformationSearch::unit [private]`

The nucleotide unit for the first nucleotide in the strand. Used for searching for conformations.

6.6.4.23   verbose_   `bool PNAB::ConformationSearch::verbose_ [private]`

Whether to print progress report to screen.

The documentation for this class was generated from the following files:

- ConformationSearch.h
- ConformationSearch.cpp

6.7   PNAB::ConformerData Struct Reference

Class to contain important information for an individual conformer.

```
#include <Containers.h>
```

Public Member Functions

- bool operator< (const ConformerData &cd) const
    Used for simple sorting based on total energy of the conformer.

Public Attributes

- OpenBabel::OBMol molecule

    The openbabel OBMol object for the conformer.

- double ∗ monomer_coord

    Pointer to array containing coordinates of a single monomer.

- double distance

    distance between interconnects in Backbone for adjacent BaseUnit

- double bondE

    Energy of newly formed bonds in the backbone divided by the length of the strand -1.

- double angleE

    Energy of newly formed angles in the backbone divided by the length of the strand -1.

- double torsionE

    Energy of all rotatable torsions divided by the length of the strand.

- double VDWE

    Total van Der Wals Energy divided by the length of the strand.

- double total_energy

    Total energy of the conformation divided by divided by the length of the strand.

- double rmsd

    Root-mean square distance relative to lowest energy conformer.

- std::size_t index

    The index of the conformer.

- bool accepted

    Is the energy of the conformer less than the thresholds.

- std::vector< double > dihedral_angles

    The values of the rotatable dihedral angles in the conformer in degrees.

6.7.1 Detailed Description

Class to contain important information for an individual conformer.

It includes detailed information about the energy components important for distinguishing between different conformers. It also includes the value of the RMSD relative to the best candidate. If the conformer satisfies the distance and energy thresholds, then it is saved. It also contains the openbabel OBMol object for the accepted candidates

See also

> Chain::generateConformerData
>
> ConformationSearch::reportData
>
> RuntimeParameters::energy_filter
>
> RuntimeParameters::max_distance

6.7.2 Member Function Documentation

6.7.2.1 operator<() `bool PNAB::ConformerData::operator< (`
`            const ConformerData & cd ) const  [inline]`

Used for simple sorting based on total energy of the conformer.

Parameters

cd	The ConformerData element to compare current element to

Returns

True if the other ConformerData has greater total energy, false otherwise

6.7.3 Member Data Documentation

6.7.3.1 accepted `bool PNAB::ConformerData::accepted`

Is the energy of the conformer less than the thresholds.

6.7.3.2 angleE `double PNAB::ConformerData::angleE`

Energy of newly formed angles in the backbone divided by the length of the strand -1.

6.7.3.3 bondE `double PNAB::ConformerData::bondE`

Energy of newly formed bonds in the backbone divided by the length of the strand -1.

6.7.3.4 dihedral_angles `std::vector<double> PNAB::ConformerData::dihedral_angles`

The values of the rotatable dihedral angles in the conformer in degrees.

6.7.3.5 distance `double PNAB::ConformerData::distance`

distance between interconnects in Backbone for adjacent BaseUnit

6.7.3.6 index `std::size_t PNAB::ConformerData::index`

The index of the conformer.

6.7.3.7 molecule `OpenBabel::OBMol PNAB::ConformerData::molecule`

The openbabel OBMol object for the conformer.

6.7.3.8 monomer_coord `double* PNAB::ConformerData::monomer_coord`

Pointer to array containing coordinates of a single monomer.

6.7.3.9 rmsd `double PNAB::ConformerData::rmsd`

Root-mean square distance relative to lowest energy conformer.

6.7.3.10 torsionE `double PNAB::ConformerData::torsionE`

Energy of all rotatable torsions divided by the length of the strand.

6.7.3.11 total_energy `double PNAB::ConformerData::total_energy`

Total energy of the conformation divided by divided by the length of the strand.

6.7.3.12 VDWE `double PNAB::ConformerData::VDWE`

Total van Der Wals Energy divided by the length of the strand.

The documentation for this struct was generated from the following file:

- Containers.h

6.8 PNAB::HelicalParameters Class Reference

A class for holding values for all helical parameters.

`#include <Containers.h>`

Public Member Functions

- [HelicalParameters](#) ()

    Empty constructor.
- void [computeHelicalParameters](#) ()

    A function to compute the helical parameters. This should be called when the the step parameters are provided.
- OpenBabel::vector3 [getGlobalTranslationVec](#) (bool is_base_pair=false, bool is_second_strand=false)

    Get the global translation vector.
- OpenBabel::vector3 [getStepTranslationVec](#) (unsigned n=0, bool is_base_pair=false, bool is_second_↩
strand=false)

    Get the step translation vector.
- OpenBabel::matrix3x3 [getGlobalRotationMatrix](#) (bool is_base_pair=false, bool is_second_strand=false)

    Get the global rotation matrix.
- OpenBabel::matrix3x3 [getStepRotationMatrix](#) (unsigned n=0, bool is_base_pair=false, bool is_second_↩
strand=false)

    Get the step rotation matrix.

Public Attributes

- double [inclination](#)

    Inclination.
- double [tip](#)

    Tip.
- double [h_twist](#)

    Helical twist.
- double [x_displacement](#)

    X-Displacement.
- double [y_displacement](#)

    Y-Displacement.
- double [h_rise](#)

    Helical rise.
- double [shift](#)

    Shift.
- double [slide](#)

    Slide.
- double [rise](#)

    Rise.
- double [tilt](#)

    Tilt.
- double [roll](#)

    Roll.
- double [twist](#)

    Twist.
- double [buckle](#)

    Buckle.
- double [propeller](#)

    Propeller twist.
- double [opening](#)

    Opening.
- double [shear](#)

    Shear.

- double stretch

    Stretch.

- double stagger

    Stagger.

- bool is_helical

    Are the base parameters helical or step parameters.

Private Member Functions

- OpenBabel::matrix3x3 rodrigues_formula (OpenBabel::vector3 axis_vector, double theta)

    Rodrigues rotation formula for rotating a vector in space.

- std::vector< OpenBabel::vector3 > StepParametersToReferenceFrame ()

    Computes the origin and direction vectors given a set of step parameters.

- void ReferenceFrameToHelicalParameters (OpenBabel::vector3 origin2, OpenBabel::vector3 x2, Open←
  Babel::vector3 y2, OpenBabel::vector3 z2)

    Computes the helical parameters given the origin and direction vectors of the second base.

6.8.1   Detailed Description

A class for holding values for all helical parameters.

The helical parameters are used for generating the geometries of the nucleobases in the strands. This class holds
the value for six helical parameters (helical twist, inclination, tip, helical rise, x-displacement, and y-displacement).
It can also compute the helical parameters given the step parameters (twist, roll, tilt, rise, slide, shift). Internally,
when the step parameters are provided, the helical parameters are computed. The helical parameters are used for
generating the geometries. This class also implements the base-pair parameters (buckle, propeller, opening, shear,
stretch, and stagger). The class holds functions to generate the geometries.

See also

> Chain::setCoordsForChain
>
> ConformationSearch::measureDistance

6.8.2   Constructor & Destructor Documentation

6.8.2.1   HelicalParameters()   `PNAB::HelicalParameters::HelicalParameters ( ) [inline]`

Empty constructor.

This empty constructor can be used. After that, values for the member variables should be specified.

6.8.3   Member Function Documentation

6.8.3.1 computeHelicalParameters() `void HelicalParameters::computeHelicalParameters ( )`

A function to compute the helical parameters. This should be called when the the step parameters are provided.

See also

is_helical

StepParametersToReferenceFrame

ReferenceFrameToHelicalParameters

6.8.3.2 getGlobalRotationMatrix() `matrix3x3 HelicalParameters::getGlobalRotationMatrix (`
`            bool is_base_pair = false,`
`            bool is_second_strand = false )`

Get the global rotation matrix.

For the base step transformation, the HelicalParameters::tip and HelicalParameters::inclination are used. For the base-pair transformation, the HelicalParameters::buckle and HelicalParameters::propeller are used. The base pair parameters are divided by two and set to positive or negative values depending on which strand is being constructed.

Parameters

is_base_pair	Whetehr the vector requested is for base-pair transformation
is_second_strand	Whether the strand being constructed is the second strand. Used only for base-pair parameters

Returns

The global rotation matrix

See also

rodrigues_formula

6.8.3.3 getGlobalTranslationVec() `vector3 HelicalParameters::getGlobalTranslationVec (`
`            bool is_base_pair = false,`
`            bool is_second_strand = false )`

Get the global translation vector.

For the base step transformation, the HelicalParameters::x_displacement and HelicalParameters::y_displacement are used. For the base-pair transformation, the HelicalParameters::shear and HelicalParameters::stretch are used. The base pair parameters are divided by two and set to positive or negative values depending on which strand is being constructed.

Parameters

is_base_pair	Whetehr the vector requested is for base-pair transformation
is_second_strand	Whether the strand being constructed is the second strand. Used only for base-pair parameters

Returns

The translation vector

6.8.3.4   getStepRotationMatrix() `matrix3x3 HelicalParameters::getStepRotationMatrix (`
`             unsigned n = 0,`
`             bool is_base_pair = false,`
`             bool is_second_strand = false )`

Get the step rotation matrix.

For the base step transformation, the HelicalParameters::twist is used. For the base-pair transformation, the HelicalParameters::opening is used. The base pair parameters are divided by two and set to positive or negative values depending on which strand is being constructed.

Parameters

n	The sequence of the nucleobase in the strand. Used only for the base step transformation
is_base_pair	Whetehr the vector requested is for base-pair transformation
is_second_strand	Whether the strand being constructed is the second strand. Used only for base-pair parameters

Returns

The step rotation matrix

6.8.3.5   getStepTranslationVec() `vector3 HelicalParameters::getStepTranslationVec (`
`             unsigned n = 0,`
`             bool is_base_pair = false,`
`             bool is_second_strand = false )`

Get the step translation vector.

For the base step transformation, the HelicalParameters::h_rise is used. For the base-pair transformation, the HelicalParameters::stagger is used. The base pair parameters are divided by two and set to positive or negative values depending on which strand is being constructed.

Parameters

n	The sequence of the nucleobase in the strand. Used only for the base step transformation
is_base_pair	Whetehr the vector requested is for base-pair transformation
is_second_strand	Whether the strand being constructed is the second strand. Used only for base-pair

| parameters |

Returns

> The step translation vector

6.8.3.6 ReferenceFrameToHelicalParameters() `void HelicalParameters::ReferenceFrameToHelical↩`
`Parameters (`
            `OpenBabel::vector3 origin2,`
            `OpenBabel::vector3 x2,`
            `OpenBabel::vector3 y2,`
            `OpenBabel::vector3 z2 ) [private]`

Computes the helical parameters given the origin and direction vectors of the second base.

This function computes the helical parameters and set the values for the appropriate member variables

Reference: Lu, Xiang-Jun, M. A. El Hassan, and C. A. Hunter. "Structure and conformation of helical nucleic acids: analysis program (SCHNAaP)." Journal of molecular biology 273.3 (1997): 668-680.

Parameters

origin2	The coordinates of the origin of the second base pair
x2	The x direction vector for the second base pair
y2	The y direction vector for the second base pair
z2	The z direction vector for the second base pair

See also

> [StepParametersToReferenceFrame](#)

6.8.3.7 rodrigues_formula() `matrix3x3 HelicalParameters::rodrigues_formula (`
            `OpenBabel::vector3 axis_vector,`
            `double theta ) [private]`

Rodrigues rotation formula for rotating a vector in space.

Outputs a 3x3 rotation matrix.

Parameters

axis_vector	A unit vector defining the axis about which to rotate by angle theta
theta	The angle at which to rotate about vector given by axis

Returns

> The new rotation matrix

6.8.3.8 StepParametersToReferenceFrame() `vector< vector3 > HelicalParameters::StepParameters↩`
`ToReferenceFrame ( )  [private]`

Computes the origin and direction vectors given a set of step parameters.

Outputs a list containing the new origin, and x, y, and z direction vectors. It uses the class member variables to access the step parameters.

Reference: El Hassan, M. A., and C. R. Calladine. "The assessment of the geometry of dinucleotide steps in double-helical DNA; a new local calculation scheme." Journal of molecular biology 251.5 (1995): 648-664.

Returns

The origin and direction vectors

See also

ReferenceFrameToHelicalParameters

6.8.4 Member Data Documentation

6.8.4.1 buckle `double PNAB::HelicalParameters::buckle`

Buckle.

6.8.4.2 h_rise `double PNAB::HelicalParameters::h_rise`

Helical rise.

6.8.4.3 h_twist `double PNAB::HelicalParameters::h_twist`

Helical twist.

6.8.4.4 inclination `double PNAB::HelicalParameters::inclination`

Inclination.

6.8.4.5 is_helical `bool PNAB::HelicalParameters::is_helical`

Are the base parameters helical or step parameters.

6.8.4.6 opening `double PNAB::HelicalParameters::opening`

Opening.

6.8.4.7 propeller `double PNAB::HelicalParameters::propeller`

Propeller twist.

6.8.4.8 rise `double PNAB::HelicalParameters::rise`

Rise.

6.8.4.9 roll `double PNAB::HelicalParameters::roll`

Roll.

6.8.4.10 shear `double PNAB::HelicalParameters::shear`

Shear.

6.8.4.11 shift `double PNAB::HelicalParameters::shift`

Shift.

6.8.4.12 slide `double PNAB::HelicalParameters::slide`

Slide.

6.8.4.13 stagger `double PNAB::HelicalParameters::stagger`

Stagger.

6.8.4.14 stretch `double PNAB::HelicalParameters::stretch`

Stretch.

6.8.4.15 tilt `double PNAB::HelicalParameters::tilt`

Tilt.

6.8.4.16 tip `double PNAB::HelicalParameters::tip`

Tip.

6.8.4.17 twist `double PNAB::HelicalParameters::twist`

Twist.

6.8.4.18 x_displacement `double PNAB::HelicalParameters::x_displacement`

X-Displacement.

6.8.4.19 y_displacement `double PNAB::HelicalParameters::y_displacement`

Y-Displacement.

The documentation for this class was generated from the following files:

- Containers.h
- Containers.cpp

6.9 driver.pNAB Class Reference

The proto-Nucleic Acid Builder main python class.

Inheritance diagram for driver.pNAB:
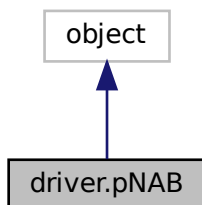
Collaboration diagram for driver.pNAB:
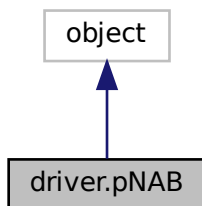
Public Member Functions

- def __init__ (self, file_path)

    The constructor for the pNAB run.

- def run (self, number_of_cpus=None, verbose=True, interrupt=False)

    Prepare helical configurations and run them in parallel.

Public Attributes

- options

    Validated options dictionary for the pNAB run.

- header

    A string comma-separated header for the results used in the generated CSV files.

- prefix

    A dictionary of the sequence of the run and the corresponding helical configuration For runs that have ranges of values for the helical parameters, this dictionary provides a mapping between the sequence of the run and the helical parameters.

- results

    A numpy array containing the results of all the candidates.

Private Member Functions

- def _run (self, config)

    Function to run one helical configuration.

Private Attributes

- _options
- _verbose

    Whether to print progress report to the screen.

- _is_helical

6.9.1 Detailed Description

The proto-Nucleic Acid Builder main python class.

A class that contains methods to create a pNAB run. It validates input, runs it, and save results.

Examples: Use the code to get RNA and DNA candidates. This uses the provided example files in the "pnab/data" directory. The results are in three files:

1) results.csv: Contains all the results for all the helical configurations requested in the run.

2) prefix.yaml: Containts a dictionary of the sequence of the run and the corresponding helical configuration (useful when a range of values is given for the helical parameters).

```
import pnab
run = pnab.pNAB('RNA.yaml')
run.run()
import pnab
run = pnab.pNAB('DNA.yaml')
run.run()
```

6.9.2 Constructor & Destructor Documentation

6.9.2.1 __init__() def driver.pNAB.__init__ (
              self,
              file_path)

The constructor for the pNAB run.

The constructor takes an input file as an argument and initialize a new instance. The input file uses a YAML format for specifying the run options. This input file can be generated using the graphical user interface created in *jupyter_widgets*. Additionally, there are a few example files in the "pnab/data" directory. The constructor creates the *pNAB.pNAB.options* attribute which contains a dictionary of all the defined options.

Parameters

file_path	(str) Path to a file containing the user-defined input. This function tries to find the file in the current working directory. If it does not exist, it looks for the file in the "pnab/data" directory

See also

jupyter_widgets

options.validate_all_options

6.9.3 Member Function Documentation

6.9.3.1 _run() `def driver.pNAB._run (`
        `self,`
        `config )` `[private]`

Function to run one helical configuration.

Setup the options to call the C++ code using the pybind11 bind class in *binder.cpp*. After the run finishes, it writes the results. Two files are written: "prefix.yaml" contains a dictionary of the sequence of the run and the corresponding helical configuration "results.csv" contains the information on the accepted candidates for all of configurations

Parameters

config	(list) a list containing two entries: the list of helical parameter values for this configuration, and a string index for the sequnce of the run

See also

run

6.9.3.2 run() `def driver.pNAB.run (`
        `self,`
        `number_of_cpus = None,`
        `verbose = True,`
        `interrupt = False )`

Prepare helical configurations and run them in parallel.

This function first copies the user-defined options to an internal options dictionary (self._options). Next, self._options is validated through a call to *options.validate_all_options* to validate all the options for the backbone parameters, helical parameters, and runtime parameters. Then, it adds the library of all the defined nucleobases. The nucleobases are defined in the "pnab/data" directory. Whether to pair adenine with thymine (default) or uracil is determined here by the *pNAB.pNAB.options*['RuntimeParameters']['pair_A_U'].

If a single value is given for a helical parameter (e.g. helical twist), then that value is used. If a range of values is given, then equally spaced values in the range will be used. The number of configurations is determined by the third value in *pNAB.pNAB.options*. The various helical configurations are run in parallel using the multiprocessing library.

This function writes three output files: "results.csv" and "prefix.yaml". It renames any existing files with these names by prepending enough "_".

Parameters

number_of_cpus	Number of CPUs to use for parallel computations of different helical configurations, defaults to all cores
verbose	Whether to print progress report to the screen, default to True
interrupt	How to handle keyboard interrupt in multiprocessing

Returns

      None; output files are written

6.9.4 Member Data Documentation

6.9.4.1 _is_helical `driver.pNAB._is_helical` `[private]`

6.9.4.2 _options `driver.pNAB._options` `[private]`

6.9.4.3 _verbose `driver.pNAB._verbose` `[private]`

Whether to print progress report to the screen.

6.9.4.4 header `driver.pNAB.header`

A string comma-separated header for the results used in the generated CSV files.

6.9.4.5 options `driver.pNAB.options`

Validated options dictionary for the [pNAB](#) run.

See also

      [options.validate_all_options](#)

6.9.4.6 prefix `driver.pNAB.prefix`

A dictionary of the sequence of the run and the corresponding helical configuration For runs that have ranges of values for the helical parameters, this dictionary provides a mapping between the sequence of the run and the helical parameters.

6.9.4.7 results `driver.pNAB.results`

A numpy array containing the results of all the candidates.

The columns in the array correspond to the entries in *pNAB.pNAB.header* and the helical configurations correspond to those in *pNAB.pNAB.prefix*

The documentation for this class was generated from the following file:

- driver.py

6.10 PNAB::RuntimeParameters Class Reference

A class for holding necessary and optional runtime parameters for conformational searches.

```
#include <Containers.h>
```

Public Member Functions

- RuntimeParameters ()

    Empty constructor.

Public Attributes

- std::vector< double > energy_filter

    [max bond E, max angle E, max torsion E, max VDW E, max total E]
- double max_distance

    Maximum accepted distance (Angstrom) between head and tail of successive nucleotides.
- std::string ff_type

    The type of the forcefield such as "GAFF" or "MMFF94"; available through Openbabel.
- std::string search_algorithm

    The search algorithm.
- std::size_t num_steps

    The number of points sampled in Monte Carlo and random searches and the number of generations in the genetic algorithm search.
- unsigned int seed
- double dihedral_step

    The dihedral step size for systematic search (degrees)
- double weighting_temperature

    The temperature used to compute the weighted probability for weighted Monte Carlo and weighted random searches.
- double monte_carlo_temperature

    The temperature used in the Monte Carlo acceptance and rejection procedure.

- double mutation_rate

    The mutation rate in the genetic algorithm search.

- double crossover_rate

    The crossover rate in the genetic algorithm search.

- int population_size

    The population size in the genetic algorithm search.

- std::vector< std::string > strand

    The names of each base used in the strand.

- std::vector< bool > build_strand

    Defines whether to build a given strand.

- std::vector< bool > strand_orientation

    Defines strand orientation for each strand in the hexad.

- bool is_hexad

    Defines whether the 60 degrees rotation for hexads is performed.

- double glycosidic_bond_distance

    Set a user-defined glycosidic bond distance (in Angstroms). If zero (default), sets the distance based on van der Waals radii.

- unsigned int num_candidates

    Quit after finding the specified number of accepted candidates.

6.10.1   Detailed Description

A class for holding necessary and optional runtime parameters for conformational searches.

The runtime parameters are used for building the strands and during the conformational search.

See also

> Chain
>
> ConformationSearch

6.10.2   Constructor & Destructor Documentation

6.10.2.1   **RuntimeParameters()**   `PNAB::RuntimeParameters::RuntimeParameters ( )  [inline]`

Empty constructor.

This empty constructor can be used. After that, values for the member variables should be specified.

6.10.3   Member Data Documentation

6.10.3.1 build_strand `std::vector<bool> PNAB::RuntimeParameters::build_strand`

Defines whether to build a given strand.

See also

Chain::Chain

6.10.3.2 crossover_rate `double PNAB::RuntimeParameters::crossover_rate`

The crossover rate in the genetic algorithm search.

See also

ConformationSearch::GeneticAlgorithmSearch

6.10.3.3 dihedral_step `double PNAB::RuntimeParameters::dihedral_step`

The dihedral step size for systematic search (degrees)

The number of steps will be $(\frac{360.0}{\text{dihedral step}})^{\text{number of rotatable dihedrals}}$.

See also

ConformationSearch::SystematicSearch

6.10.3.4 energy_filter `std::vector<double> PNAB::RuntimeParameters::energy_filter`

[max bond E, max angle E, max torsion E, max VDW E, max total E]

```
- Maximum accepted energy for newly formed bonds in the backbone (kcal/mol/bond)
- Maximum accepted energy for newly formed angles in the backbone (kcal/mol/angle)
- Maximum accepted torsional energy for rotatable bonds (kcal/mol/nucleotide)
- Maximum accepted van der Waals energy (kcal/mol/nucleotide)
- Maximum accepted total energy (kcal/mol/nucleotide)

@sa Chain::generateConformerData
@sa Chain::fillConformerEnergyData
```

6.10.3.5   ff_type `std::string PNAB::RuntimeParameters::ff_type`

The type of the forcefield such as "GAFF" or "MMFF94"; available through Openbabel.

See also

    Chain::Chain

6.10.3.6   glycosidic_bond_distance `double PNAB::RuntimeParameters::glycosidic_bond_distance`

Set a user-defined glycosidic bond distance (in Angstroms). If zero (default), sets the distance based on van der Waals radii.

6.10.3.7   is_hexad `bool PNAB::RuntimeParameters::is_hexad`

Defines whether the 60 degrees rotation for hexads is performed.

6.10.3.8   max_distance `double PNAB::RuntimeParameters::max_distance`

Maximum accepted distance (Angstrom) between head and tail of successive nucleotides.

This distance is used to quickly screen rotamers. The algorithm searches for backbone candidates that are periodic in terms of the helical structure. Thus, the terminal atoms in adjacent candidates must be within a small distance to allow for a bond to form. This distance is recommended to be less than 0.1 Angstroms. The extra terminal atom in the adjacent nucleotide is then removed.

See also

    ConformationSearch::measureDistance

6.10.3.9   monte_carlo_temperature `double PNAB::RuntimeParameters::monte_carlo_temperature`

The temperature used in the Monte Carlo acceptance and rejection procedure.

See also

    ConformationSearch::MonteCarloSearch

6.10.3.10 mutation_rate `double PNAB::RuntimeParameters::mutation_rate`

The mutation rate in the genetic algorithm search.

See also

ConformationSearch::GeneticAlgorithmSearch

6.10.3.11 num_candidates `unsigned int PNAB::RuntimeParameters::num_candidates`

Quit after finding the specified number of accepted candidates.

6.10.3.12 num_steps `std::size_t PNAB::RuntimeParameters::num_steps`

The number of points sampled in Monte Carlo and random searches and the number of generations in the genetic algorithm search.

See also

ConformationSearch::MonteCarloSearch

ConformationSearch::RandomSearch

ConformationSearch::GeneticAlgorithmSearch

6.10.3.13 population_size `int PNAB::RuntimeParameters::population_size`

The population size in the genetic algorithm search.

See also

ConformationSearch::GeneticAlgorithmSearch

6.10.3.14 search_algorithm `std::string PNAB::RuntimeParameters::search_algorithm`

The search algorithm.

There are six search algorithms:

- Systematic search

- Monte Carlo search

- Weighted Monte Carlo search

- Random search

- Weighted random search

- Genetic algorithm search

  See also

  ConformationSearch

6.10.3.15 seed `unsigned int PNAB::RuntimeParameters::seed`

6.10.3.16 strand `std::vector<std::string> PNAB::RuntimeParameters::strand`

The names of each base used in the strand.

6.10.3.17 strand_orientation `std::vector<bool> PNAB::RuntimeParameters::strand_orientation`

Defines strand orientation for each strand in the hexad.

See also

Chain::setCoordsForChain

Chain::setCoordsForChain

6.10.3.18 weighting_temperature `double PNAB::RuntimeParameters::weighting_temperature`

The temperature used to compute the weighted probability for weighted Monte Carlo and weighted random searches.

See also

ConformationSearch::WeightedDistributions

The documentation for this class was generated from the following file:

- Containers.h

7 File Documentation

7.1 advanced.dox File Reference

7.2 binder.cpp File Reference

A file for exporting classes and functions to python using pybind11.

```
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>
#include <pybind11/stl_bind.h>
#include <pybind11/iostream.h>
#include "ConformationSearch.h"
```
Include dependency graph for binder.cpp:

Namespaces

- PNAB

  The PNAB name space contains all the C++ classes and functions for the proto-Nucleic Acid Builder.

Functions

- std::string PNAB::run (PNAB::RuntimeParameters runtime_params, PNAB::Backbone &py_backbone, std←
  ::vector< PNAB::Base > py_bases, PNAB::HelicalParameters hp, std::string prefix="run", bool verbose=true)

  A wrapper function to run the search algorithm code from python.
- PNAB::PYBIND11_MODULE (bind, m)

  Exports certain classes to python to allow the user to run the code from python.

7.2.1 Detailed Description

A file for exporting classes and functions to python using pybind11.

7.3 Chain.cpp File Reference

A file for defining various methods for building and evaluating nucleic acid strands.

```
#include "Chain.h"
```
Include dependency graph for Chain.cpp:
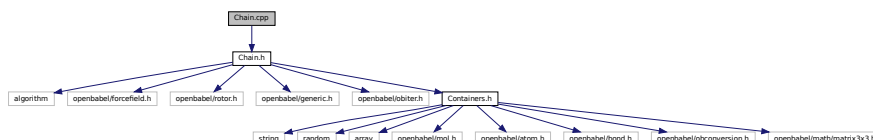
7.3.1 Detailed Description

A file for defining various methods for building and evaluating nucleic acid strands.

7.4 Chain.h File Reference

A file for declaring a class for building and evaluating nucleic acid strands.

```
#include <algorithm>
#include <openbabel/forcefield.h>
#include <openbabel/rotor.h>
#include <openbabel/generic.h>
#include <openbabel/obiter.h>
```

```
#include "Containers.h"
```
Include dependency graph for Chain.h:

This graph shows which files directly or indirectly include this file:
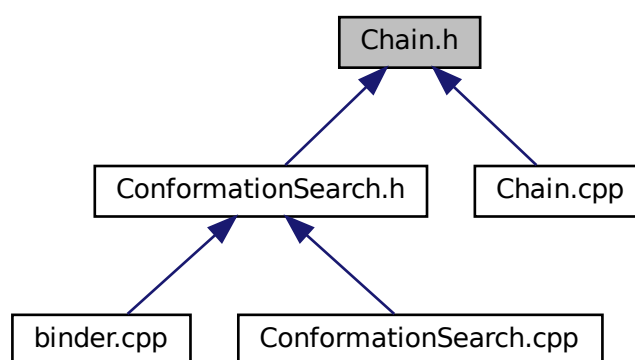
Classes

- class PNAB::Chain

  A class for building nucleic acid strands and evaluating their energies.

Namespaces

- PNAB

  The PNAB name space contains all the C++ classes and functions for the proto-Nucleic Acid Builder.

Macros

- #define KJ_TO_KCAL 0.239006

7.4.1   Detailed Description

A file for declaring a class for building and evaluating nucleic acid strands.

7.4.2 Macro Definition Documentation

7.4.2.1 KJ_TO_KCAL `#define KJ_TO_KCAL 0.239006`

7.5 ConformationSearch.cpp File Reference

A file for defining various search algorithm functions.

```
#include <iomanip>
#include <set>
#include <pybind11/pybind11.h>
#include "ConformationSearch.h"
```
Include dependency graph for ConformationSearch.cpp:
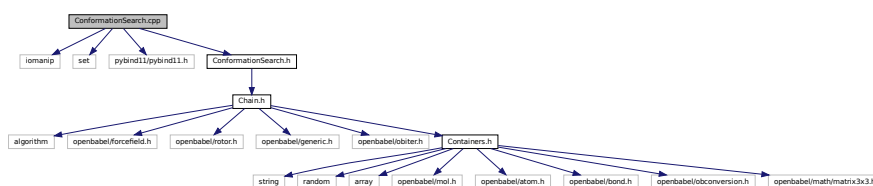
7.5.1 Detailed Description

A file for defining various search algorithm functions.

7.6 ConformationSearch.h File Reference

A file for declaring the search algorithm class.

```
#include "Chain.h"
```
Include dependency graph for ConformationSearch.h:
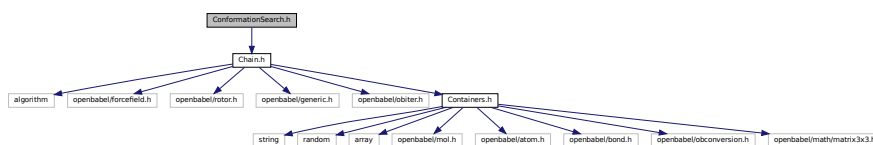
This graph shows which files directly or indirectly include this file:
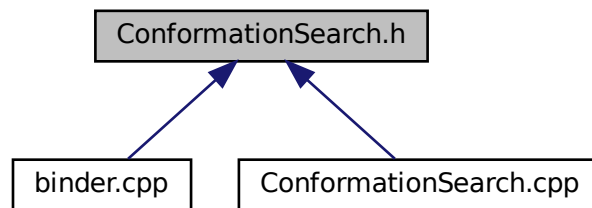
Classes

- class PNAB::ConformationSearch

    A rotor search function used to find acceptable conformations of arbitrary backbone and helical parameter combinations. The main class of the proto-Nucleic Acid Builder.

Namespaces

- PNAB

    The PNAB name space contains all the C++ classes and functions for the proto-Nucleic Acid Builder.

Macros

- #define BOLTZMANN 0.0019872041

7.6.1   Detailed Description

A file for declaring the search algorithm class.

7.6.2   Macro Definition Documentation

7.6.2.1   BOLTZMANN   `#define BOLTZMANN 0.0019872041`

7.7 Containers.cpp File Reference

A file for defining various methods for defining options.

```
#include "Containers.h"
#include <openbabel/math/align.h>
```
Include dependency graph for Containers.cpp:
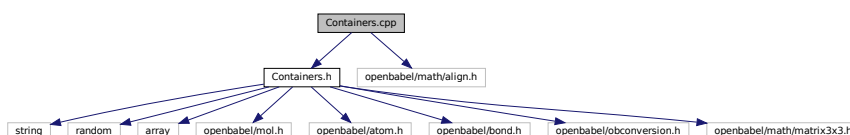
7.7.1 Detailed Description

A file for defining various methods for defining options.

7.8 Containers.h File Reference

A file for declaring various classes for defining options.

```
#include <string>
#include <random>
#include <array>
#include <openbabel/mol.h>
#include <openbabel/atom.h>
#include <openbabel/bond.h>
#include <openbabel/obconversion.h>
#include <openbabel/math/matrix3x3.h>
```
Include dependency graph for Containers.h:
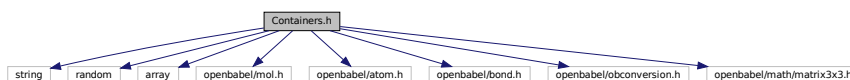
This graph shows which files directly or indirectly include this file:
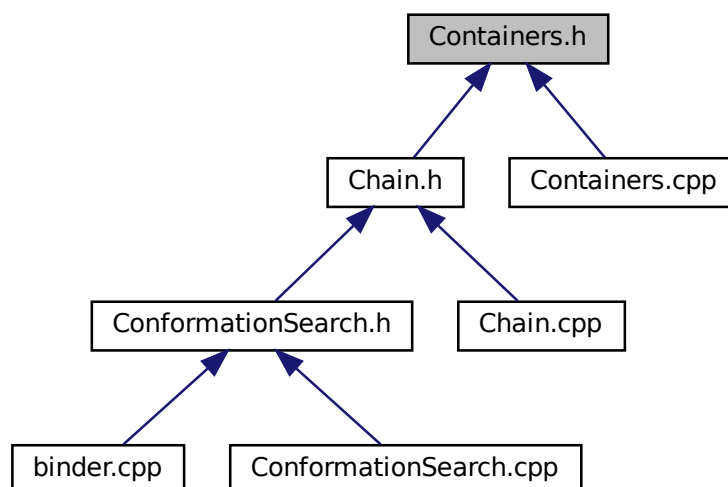
Classes

- class PNAB::RuntimeParameters

    A class for holding necessary and optional runtime parameters for conformational searches.

- class PNAB::HelicalParameters

    A class for holding values for all helical parameters.

- class PNAB::Backbone

    Class for holding backbone information.

- class PNAB::Base

    Class to fully define bases (i.e. Adenine, Cytosine)

- class PNAB::Bases

    A class that contains a vector of all the defined bases and a funtion to return a base and the complimentary base for all the bases defined.

- class PNAB::BaseUnit

    Class to hold bases with backbones attached (nucleotides), along with associated necessary information.

- struct PNAB::ConformerData

    Class to contain important information for an individual conformer.

Namespaces

- PNAB

    The PNAB name space contains all the C++ classes and functions for the proto-Nucleic Acid Builder.

Macros

- #define M_PI 3.14159265358979323846
- #define DEG_TO_RAD (M_PI/180.0)

7.8.1 Detailed Description

A file for declaring various classes for defining options.

7.8.2 Macro Definition Documentation

7.8.2.1 DEG_TO_RAD `#define DEG_TO_RAD (`M_PI`/180.0)`

7.8.2.2 M_PI `#define M_PI 3.14159265358979323846`

7.9 driver.py File Reference

Classes

- class driver.pNAB

    The proto-Nucleic Acid Builder main python class.

Namespaces

- driver

    This is the main file for the pnab driver.

Functions

- def driver.init_worker ()

Variables

- driver.category

7.9.1 Detailed Description

This is the main file for the pnab driver

7.10 jupyter_widgets.py File Reference

Namespaces

- jupyter_widgets

    A file for displaying widgets in the Jupyter notebook.

Functions

- def jupyter_widgets.view_nglview (molecule, label=False)

  Display molecules using the NGLView viewer.
- def jupyter_widgets.fixed_bonds (num_bonds, num_atoms, param)

  Display widgets for determining indices of fixed bonds.
- def jupyter_widgets.path (file_path, param)

  Display backbone to Jupyter notebook given a file path.
- def jupyter_widgets.upload_backbone (f, param)

  Upload a backbone file to Jupyter notebook.
- def jupyter_widgets.backbone (param)

  Main backbone widget for use in Jupyter notebook.
- def jupyter_widgets.base_path (file_path, param, base_number)

  Display base to Jupyter notebook given a file path.
- def jupyter_widgets.upload_base (f, param, base_number)

  Upload a base file to Jupyter notebook.
- def jupyter_widgets.add_base (number_of_bases, param)

  Display widgets to upload the requested number of bases.
- def jupyter_widgets.bases (param)

  Bases widget for use in Jupyter notebook.
- def jupyter_widgets.helical_parameters (param)
- def jupyter_widgets.algorithm (chosen_algorithm, param)

  Display search parameters based on the chosen algorithm.
- def jupyter_widgets.runtime_parameters (param)

  Runtime parameter widget for use in Jupyter notebook.
- def jupyter_widgets.upload_input (param, f)

  Widget to upload an input file.
- def jupyter_widgets.display_options_widgets (param, input_file, uploaded=False)

  Display all widgets in Jupyter notebook given an input file.
- def jupyter_widgets.user_input_file (param)

  Display input file options.
- def jupyter_widgets.run (button)

  Function to run the code when the user finishes specifying all options.
- def jupyter_widgets.extract_options ()

  Extracts user options from the widgets.
- def jupyter_widgets.single_result (result, header, results, prefix)

  Interactive function to display a single result.
- def jupyter_widgets.show_results (results, header, prefix)

  Display results.
- def jupyter_widgets.builder ()

  The function called from the Jupyter notebook to display the widgets.

Variables

- dictionary jupyter_widgets.input_options = {}

  Stores the widgets corresponding to the user-defined options.

7.10.1   Detailed Description

A file for displaying widgets in the Jupyter notebook

7.11 mainpage.dox File Reference

7.12 options.py File Reference

Namespaces

- options

    A file for preparing, explaining and validating options.

Functions

- def options.validate_all_options (options)

    A method to validate all options.
- def options._align_nucleobase (base_options)

    Aligns the provided nucleobase to purine or pyrimidine in the nucleic acid base pair standard reference frame.
- def options._validate_input_file (file_name)

    Method to validate that the given file exists.
- def options._validate_atom_indices (indices)

    Method to validate provided lists of atom indices.
- def options._validate_helical_parameters (hp_i)

    Method to validate provided helical parameters for each parameter.
- def options._validate_energy_filter (energy_filter)

    Method to validate provided energy filter.
- def options._validate_strand (strand)

    Method to validate provided strand sequence.
- def options._validate_bool_list (x)

    Method to validate a list of bools.

Variables

- dictionary options._options_dict = {}

    Nucleic acid builder options used in the driver.

7.12.1 Detailed Description

A file for preparing, explaining and validating options

Index