Codon Harmony Documentation

Release 1.0.0

Brian D. Weitzner

Mar 20, 2019

Contents:

CHAPTER 1

Codon Harmony

1.1 Features

1. Reverse translates input amino acid sequence to DNA.
2. Calculates the host's per-AA codon usage profile – codons used less than a specified threshold (defaults to 10%) are dropped.
3. Compares the reverse-translated DNA sequence to the host profile, determines which codons are overused/underused.
4. Stochastically mutates codons according to host profile.
5. Ranks sequences by codon adaptation index relative to host
6. Processes DNA to remove unwanted features:
    - high GC content within a sliding window and across the entire sequence
    - unwanted restriction sites
    - alternate start positions (GA-rich regions 18 bp upstream of ATG/GTG/TTG)
    - 3-consecutive identical codons and 9-mer repeat chunks
    - areas with more than 4 (variable) consecutive identical bps ("local homopolymers")
    - RNA hairpins, detected by looking for 10-mers with reverse complements (including wobble bases) in the sequence
    - RNA splice sites, detected by similarity to consensus donor and acceptor site sequences

The process is repeated from step 3 for a specified number of cycles (defaults to 1000) OR until the per-AA codon profile of current DNA and host profile matches (within tolerance).

1.2 Future work

- More advanced RNA-structure removal
  - CONTRAfold – overkill for now
  - nupack – overkill for now

CHAPTER 2

Installation

2.1 Stable release

To install Codon Harmony, run this command in your terminal:

```
$ pip install codon_harmony
```

This is the preferred method to install Codon Harmony, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Codon Harmony can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/weitzner/codon_harmony
```

Or download the tarball:

```
$ curl  -OL https://github.com/weitzner/codon_harmony/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

Reverse translate your amino acid sequence harmoniously with a host's condon usage.

```
usage: codon_harmony [-h] --input INPUT --output OUTPUT [--host HOST]
                     [--host-threshold HOST_THRESHOLD]
                     [--local-host-profile LOCAL_HOST_PROFILE]
                     [--verbose {0,1,2,3}]
                     [--local-homopolymer-threshold LOCAL_HOMOPOLYMER_THRESHOLD]
                     [--cycles CYCLES] [--inner-cycles INNER_CYCLES]
                     [--max-relax MAX_RELAX]
                     [--restriction-enzymes [RESTRICTION_ENZYMES [RESTRICTION_ENZYMES␣
↪...]]]
                     [--remove-splice-sites | --no-remove-splice-sites]
                     [--remove-start-sites | --no-remove-start-sites]
```

3.1 Named Arguments

--input input file with sequence

--output output file to write DNA sequence(s)

--host host table code: http://www.kazusa.or.jp/codon/, default is "Escherichia coli B"

Default: "413997"

--host-threshold lowest codon fraction per AA in the host that is allowed

Default: 0.10

--local-host-profile path to host codon usage table as JSON file

--verbose Possible choices: 0, 1, 2, 3

verbose output level (0=only result, 1=standard output, 2=extra output 3=debugging)

Default: 0

--local-homopolymer-threshold number of consecutive NT repeats allowed

Default: 4

--cycles number of independent codon samples to run. 0 means 1 pass

Default: 10

--inner-cycles number of times to iteratively optimize each independent codon sample. 0 means 1 pass

Default: 10

--max-relax maximum percent deviation from host profile

Default: 0.1

--restriction-enzymes list of restriction enzyme sites to remove (e.g. –restriction_enzymes NdeI XhoI HpaI).

Default: ['NdeI', 'XhoI', 'HpaI', 'PstI', 'EcoRV', 'NcoI', 'BamHI']

--remove-splice-sites Remove splice sites. Use for mammalian hosts.

Default: True

--no-remove-splice-sites Do not remove splice sites.

Default: True

--remove-start-sites Remove alternate start sites. Use for bacterial hosts.

Default: True

--no-remove-start-sites Do not remove alternate start sites.

Default: True

v1.0.0 (contact bweitzner@lyellbio.com if you encounter errors)

3.2 Executing Codon Harmony as a script

python codon_harmony/codon_harmony.py –input misc/INPUT_LIST.fasta –output out.fasta

To get started, create a conda environment from the `environment.yml` file:

```
conda env create -f environment.yml
```

contents of `misc/INPUT_LIST.fasta`:

```
>test_sequence1|can be optimized with `max_relax` set to 0.1
HHHHHHHHHH
>test_sequence2|cannot be optimized with `max_relax` set to 0.1
ACDEFGHIKLMNPQRSTVWY
>test_sequence3|can be optimized with `max_relax` set to 0.1, has extreme GC content
FFFFFFFFFFF
```

3.3 Using Codon Harmony in a project

```
import codon_harmony
codon_harmony.runner()
```

The `runner` function will handle parsing all command line arguments.

CHAPTER 4

codon_harmony

4.1 codon_harmony package

4.1.1 Subpackages

codon_harmony.data module

class codon_harmony.data.**GCParams**
High and low values for GC-content within a specified window size.

name
Name of the parameter set.

Type str

window_size
Number of nucleotides over which the GC content will be calculated.

Type int

low
The minimum fraction of GC in the window.

Type float

high
The maximum fraction of GC in the window.

Type float

codon_harmony.data.**RestrictionEnzymes**(*restriction_enzymes*)
Create a RestrictionBatch instance to search for sites for a supplied list of restriction enzymes.

Parameters **restriction_enzymes** (*list[str], optional*) – List of restriction enzymes to consider. Defaults to ["NdeI", "XhoI", "HpaI", "PstI", "EcoRV", "NcoI", "BamHI"].

Returns RestrictionBatch instance configured with the input restriction enzymes.

Return type Bio.Restriction.Restriction.RestrictionBatch

codon_harmony.data.**codon_tables**(*taxid*, *table_path=None*)
Download the codon use table for the given species and return it as a dictionary.

Returns The NCBI taxonomy ID for the supplied species.

Return type int

Parameters

- **taxid** (*int*) – NCBI taxonomy ID for the desrired species.
- **table_path** (*str*) – Defaults to None. Path to a JSON-formatted file representing the codon usage to consider. If None, the table is fetched from the internet.

Raises

- ValueError – If the NCBI taxonomy ID is not associated with a codon
- usage table, raise a ValueError informing the user and directing
- them to the NCBI Taxonomy Browser.

Returns A dictionary with codons as keys and the frequency that the codon is used to encode its amino acid as values.

Return type dict{str, float}

codon_harmony.util package

codon_harmony.util.codon_use module

codon_harmony.util.codon_use.**calc_codon_relative_adaptiveness**(*codons_count*)
Calculate the relative adaptiveness of each synonymous codon from an input dictionary of counts.

Note: The claculation and some nomenclature is taken from Sharp and Li (Nucleic Acids Res. 1987 Feb 11;15(3):1281-95).

Parameters **codons_count** (*dict{str, int}*) – A dictionary with codons as keys and the corresponding number of occurences as values.

Returns A CodonAdaptationIndex instance configured to calculate CAI for a target gene.

Return type Bio.SeqUtils.CodonUsage.CodonAdaptationIndex

codon_harmony.util.codon_use.**calc_profile**(*codons_count*)
Calculate the frequency of usage of each synonymous codon from an input dictionary of counts.

Parameters **codons_count** (*dict{str, int}*) – A dictionary with codons as keys and the corresponding number of occurences as values.

Returns A dictionary with codons as keys and the corresponding frequency of occurences as values.

Return type dict{str, int}

codon_harmony.util.codon_use.**count_codons**(*dna_sequence*)
Count the number of times each codon appears in a DNA sequence.

Parameters **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.

Returns A dictionary with codons as keys and the corresponding number of occurences as values.

Return type dict{str, int}

codon_harmony.util.codon_use.**host_codon_usage**(*host*, *threshold=0.1*, *table_path=None*)

Load and process the per amino acid codon usage for the desired host in accordance with the supplied threshold and configure a CodonAdaptationIndex instance to calculate CAI for a target gene.

Note: The relative adaptiveness used in the CodonAdaptationIndex is based on the filtered codon use frequencies, not the raw counts.

Parameters

- **host** (*str*) – Latin name or NCBI taxonomy ID of the host organism.
- **threshold** (*float, optional*) – Lowest fraction of codon usage to keep. Defaults to 0.10.

Returns

A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymous codon is used.

A dictionary with codons as keys and the corresponding frequency of occurences as values.

A *CodonAdaptationIndex* instance configured to calculate CAI for a target gene.

Return type dict{str, list[list, list]}, dict{str, int}, Bio.SeqUtils.CodonUsage.CodonAdaptationIndex

codon_harmony.util.codon_use.**process_host_table**(*host*, *threshold*, *table_path*)

Load the codon usage table for the desired host, filter codons with a lower occurence than the threshold, and renormalize the frequency of usage of each synonymous codon.

Parameters

- **host** (*str*) – Latin name or NCBI taxonomy ID of the host organism.
- **threshold** (*float*) – Lowest fraction of codon usage to keep.

Returns A dictionary with codons as keys and the corresponding frequency of occurences as values.

Return type dict{str, int}

codon_harmony.util.seq module

codon_harmony.util.seq.**back_translate**(*self*)

Return the DNA sequence from an amino acid sequence by creating a new Seq object. The first codon in the synonymous codons list is always chosen for each amino acid; codon optimization is required after back translation.

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> my_protein = Seq("MAIVMGR", IUPAC.protein)
>>> my_protein
```

(continues on next page)

(continued from previous page)

```
Seq('MAIVMGR', IUPACProtein())
>>> my_protein.back_translate()
Seq('ATGGCCATTGTAATGGGCCGCTG', IUPACUnambiguousDNA())
```

Trying to back-transcribe a DNA or RNA sequence raises an exception:

```
>>> messenger_rna = Seq("AUGGCCAUUGUAAUGGGCCGCUG", IUPAC.unambiguous_rna)
>>> messenger_rna.back_translate()
Traceback (most recent call last):
...
ValueError: Nucleic acids cannot be back translated!
```

codon_harmony.util.seq_opt module

codon_harmony.util.seq_opt.**compare_profiles**(*codons_count*, *host_profile*, *relax*)

Compute the deviation from the expected codon usage based on a host codon usage profile.

Note: The *relax* parameter uniformly increases the host codon usage that is used to estimate the number of times each codon should appear in the sequence. These values are rounded and then iteratively adjusted to be consistent with the length of the sequence of interest. Increasing this parameter further distorts the codon use distribution from the host.

Parameters

- **codons_count** (*dict{str, int}*) – A dictionary with each codon as keys and the number of times it appears in a gene as values.
- **host_profile** (*dict{str, foat}*) – A dictionary with each codon as keys and the frequency of its use in the host organism as values.
- **relax** (*float*) – The maximum deviation from the host profile to tolerate.

Returns

A dictionary with each codon as keys, and dictionaries of the difference between the observed and expected codon usage.

The number of mutations per residue that are needed to make the sequence match the host codon usage.

Return type dict{str, dict{str, int}}, float

codon_harmony.util.seq_opt.**gc_scan**(*dna_sequence*, *codon_use_table*, *gc*)

Scan across a sequence and replace codons to acheive a desired GC content within the window.

Note: The following fields of the *GCParams* type are used in this function:

- **window_size** (*int*) – Size of sliding window (in nucelotides) to examine for GC content. Window sizes can also be expressed as factors of the length of *dna_sequence* by passing a string that begins with "x" (e.g. "x0.5").
- **low** (*float*) – Minimum GC content in window.
- **high** (*float*) – Maximum GC content in window.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymouscodon is used.
- **gc** (GCParams) – A *namedtuple* with fields for name, window_size, minimum and maximum GC content.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**harmonize_codon_use_with_host**(*dna_sequence*, *mutation_profile*)

Adjust the codon usage in the DNA sequence to be consistent with the host profile.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **mutation_profile** (*dict{str, dict{str, int}}*) – A dictionary with each codon as keys, and dictionaries of the difference between the observed and expected codon usage.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**mutate_codon**(*codon_in*, *codon_use_table*)

Select a synonymous codon in accordance with the frequency of use in the host organism.

Parameters

- **codon_in** (*Bio.Seq.Seq*) – A single codon.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymous codon is used.

Returns A new codon.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**remove_hairpins**(*dna_sequence*, *codon_use_table*, *stem_length=10*)

Identify and remove stretches of the equence that can form hairpins.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymouscodon is used.
- **stem_length** (*int, optional*) – Length of hairpin stem to detect. Defaults to 10.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**remove_local_homopolymers**(*dna_sequence*, *codon_use_table*, *n_codons=2*, *homopolymer_threshold=4*)

Identify and remove consecutive stretches of the same nucleotides using a sliding window of a fixed number of codons.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymouscodon is used.
- **n_codons** (*int, optional*) – Size of window (in codons) to examine. Defaults to 2.
- **homopolymer_threshold** (*int*) – number of consecutive nucleotide repeats allowed. Defaults to 4.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**remove_repeating_sequences**(*dna_sequence*, *codon_use_table*, *window_size*)

Identify and remove repeating sequences of codons or groups of codons within a DNA sequence.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymouscodon is used.
- **window_size** (*int*) – Size the window (in nucleotides) to examine. Window sizes are adjusted down to the nearest multiple of 3 so windows only contain complete codons.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**remove_restriction_sites**(*dna_sequence*, *codon_use_table*, *restrict_sites*)

Identify and remove seuences recognized by a set of restriction enzymes.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymouscodon is used.
- **restrict_sites** (*Bio.Restriction.RestrictionBatch*) – RestrictionBatch instance configured with the input restriction enzymes.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**remove_splice_sites**(*dna_sequence*, *codon_use_table*)
Identify and remove RNA splice sites within a DNA sequence.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymouscodon is used.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**remove_start_sites**(*dna_sequence*, *codon_use_table*, *ribosome_binding_sites*, *table_name='Standard'*)
Identify and remove alternate start sites using a supplied set of ribosome binding sites and a codon table name.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymouscodon is used.
- **ribosome_binding_sites** (*dict{str, str}*) – A dictionary with named ribosome binding sites as keys and the corresponding sequences as values.
- **table_name** (*str, optional*) – Name of a registered NCBI table. See *Bio.Data.CodonTable.unambiguous_dna_by_name.keys()* for options. Defaults to "Standard".

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**resample_codons**(*dna_sequence*, *codon_use_table*)
Generate a new DNA sequence by swapping synonymous codons. Codons are selected in accordance with their frequency of occurrence in the host organism.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymous codon is used.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

codon_harmony.util.seq_opt.**resample_codons_and_enforce_host_profile**(*dna_sequence*, *codon_use_table*, *host_profile*, *relax*)
Generate a new DNA sequence by swapping synonymous codons. Codons are selected in accordance with their

frequency of occurrence in the host organism and adjust the codon usage in the DNA sequence to match the host profile.

Parameters

- **dna_sequence** (*Bio.Seq.Seq*) – A read-only representation of the DNA sequence.
- **codon_use_table** (*dict{str, list[list, list]}*) – A dictionary with each amino acid three-letter code as keys, and a list of two lists as values. The first list is the synonymous codons that encode the amino acid, the second is the frequency with which each synonymous codon is used.
- **host_profile** (*dict{str, foat}*) – A dictionary with each codon as keys and the frequency of its use in the host organism as values.
- **relax** (*float*) – The maximum deviation from the host profile to tolerate.

Returns A read-only representation of the new DNA sequence.

Return type Bio.Seq.Seq

4.1.2 Module contents

codon_harmony.codon_harmony.**get_parser**()

codon_harmony.codon_harmony.**main**(*argv=None*)

Read in a fasta-formatted file containing amino acid sequences and reverse translate each of them in accordance with a specified host's codon usage frequency. The DNA sequence is then processed to remove unwanted features.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/weitzner/codon_harmony/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Codon Tools could always use more documentation, whether as part of the official Codon Tools docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/weitzner/codon_harmony/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *codon_harmony* for local development.

1. Fork the *codon_harmony* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/codon_harmony.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv codon_harmony
$ cd codon_harmony/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 codon_harmony tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/weitzner/codon_harmony/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_codon_harmony
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

6.1 Development Lead

- Brian D. Weitzner <bweitzner@lyellbio.com>
- Yang Hsia <yhsia@uw.edu>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.9.2 (2019-02-06)

- First release on PyPI.

7.2 0.9.4 (2019-02-20)

- Full suite of tests added, bugs uncovered and fixed
- Adjustments to the packaging setup – actaully installable now

7.3 0.9.5 (2019-02-25)

- Adding support for RNA splice site detection and removal

7.4 0.9.6 (2019-02-28)

- Updating the way optimization failures are reported and displayed
- Parallelizing via a process pool

7.5 1.0.0 (2019-03-06)

- Added ability to use offline tables in addition to fetching from the internet
- Full suite of tests and documentation
- Tested on real-world sequences

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index