Goldilocks Documentation

Release 0.1.1

Sam Nicholls

July 07, 2016

Contents

Contents:

CHAPTER 1

Goldilocks

Locating genomic regions that are "just right".

- Documentation: http://goldilocks.readthedocs.org.

1.1 What is it?

Goldilocks is a Python package providing functionality for locating 'interesting' genomic regions for some definition of 'interesting'. You can import it to your scripts, pass it sequence data and search for subsequences that match some criteria across one or more samples.

Goldilocks was developed to support our work in the investigation of quality control for genetic sequencing. It was used to quickly locate regions on the human genome that expressed a desired level of variability, which were "just right" for later variant calling and comparison.

The package has since been made more flexible and can be used to find regions of interest based on other criteria such as GC-content, density of target k-mers, defined confidence metrics and missing nucleotides.

1.2 What can I use it for?

Given some genetic sequences (from one or more samples, comprising of one or more chromosomes), Goldilocks will shard each chromosome in to subsequences of a desired size which may or may not overlap as required. For each chromosome from each sample, each subsequence or 'region' is passed to the user's chosen strategy.

The strategy simply defines what is of interest to the user in a language that Goldilocks can understand. Goldilocks is currently packaged with the following strategies:

Strategy	Census Description
GCRatioStrategy	Calculate GC-ratio for subregions across the genome.
NucleotideCounterStrategy	Count given nucleotides for subregions across the genome.
MotifCounterStrategy	Search for one or more particular motifs of interest of any and varying size in subregions across the genome.
ReferenceConsensusStrategy	Calculate the (dis)similarity to a given reference across the genome.
PositionCounterStrategy	Given a list of base locations, calculate density of those locations over subregions across the genome.

Once all regions have been 'censused', the results may be sorted by one of four mathematical operations: *max*, *min*, *median* and *mean*. So you may be interested in subregions of your sequence(s) that feature the most missing nucleotides, or subregions that contain the mean or median number of SNPs or the lowest GC-ratio.

1.3 Why should I use it?

Goldilocks is hardly the first tool capable of calculating GC-content across a genome, or to find k-mers of interest, or SNP density, so why should you use it as part of your bioinformatics pipeline?

Whilst not the first program to be able to conduct these tasks, it is the first to be capable of doing them all together, sharing the same interfaces. Every strategy can quickly be swapped with another by changing one line of your code. Every strategy returns regions in the same format and so you need not waste time munging data to fit the rest of your pipeline.

Strategies are also customisable and extendable, those even vaguely familiar with Python should be able to construct a strategy to meet their requirements.

Goldilocks is maintained, documented and tested, rather than that hacky perl script that you inherited years ago from somebody who has now left your lab.

1.4 Requirements

To use;

- numpy
- matplotlib (for plotting)

To test;

- tox
- pytest

For coverage;

- nose
- python-coveralls

1.5 Installation

```
$ pip install goldilocks
```

1.6 Citation

Please cite us so we can continue to make useful software!

```
Nicholls, S. M., Clare, A., & Randall, J. C. (2016). Goldilocks: a tool for identifying genomic regio
```

```
@article{Nicholls01072016,
    author = {Nicholls, Samuel M. and Clare, Amanda and Randall, Joshua C.},
    title = {Goldilocks: a tool for identifying genomic regions that are 'just right'},
    volume = {32},
    number = {13},
    pages = {2047-2049},
    year = {2016},
    doi = {10.1093/bioinformatics/btw116},
    URL = {http://bioinformatics.oxfordjournals.org/content/32/13/2047.abstract},
    eprint = {http://bioinformatics.oxfordjournals.org/content/32/13/2047.full.pdf+html},
    journal = {Bioinformatics}
}
```

1.7 License

Goldilocks is distributed under the MIT license, see LICENSE.

CHAPTER 2

Installation

At the command line:

```
$ pip install goldilocks
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv goldilocks
$ pip install goldilocks
```

CHAPTER 3

Command Line Usage

Goldilocks is also packaged with a basic command line tool to demonstrate some of its capabilities and to provide access to base functionality without requiring users to author a script of their own. For more complicated queries, you'll need to import Goldilocks as a package to a script of your own. But for simple use-cases the tool might be enough for you.

3.1 Usage

Goldilocks is invoked as follows:

```
goldilocks <strategy> <sort-op> [--tracks TRACK1 [TRACK2 ...]] -l LENGTH -s STRIDE [-@ THREADS] FAIDX
```

Where a strategy is a census strategy listed as available...

```
$ goldilocks list
Available Strategies
  * gc
  * ref
  * motif
  * nuc
```

...and a sort operation is one of:

- *max*
- *min*
- *mean*
- *median*
- *none*

3.2 Example

Tabulate all regions and their associated counts of nucleotides A, C, G, T and N. Window size 100Kbp, overlap 50Kbp. Census will spawn 4 processes. Regions in table will be sorted by co-ordinate:

```
goldilocks nuc none --tracks A C G T N -l 100000 -s 50000 -@ 4 /store/ref/hs37d5.fa.fai
```

Tabulate all regions and their associated GC-content. Same parameters as previous example but table will be sorted by maximum GC-content descending:

```
goldilocks gc max -l 100000 -s 50000 -@ 4 /store/ref/hs37d5.fa.fai
```

CHAPTER 4

Basic Package Usage

The following example assumes basic Python programming experience (and that you have installed Goldilocks), skip to the end if you think you know what you're doing.

4.1 Importing

To use Goldilocks you will need to import the `goldilocks.goldilocks.Goldilocks` class and your desired census strategy (e.g. NucleotideCounterStrategy) from `goldilocks.strategies` to your script:

```
from goldilocks import Goldilocks
from goldilocks.strategies import NucleotideCounterStrategy
```

4.2 Providing Sequence Data as Dictionary

If you do not have FASTA files, the `goldilocks.goldilocks.Goldilocks` class allows you to provide sequence data in the following structure:

```
sequence_data = {
    "sample_name_or_identifier": {
        "chr_name_or_number": "my_actual_sequence",
    }
}
```

For example:

```
sequence_data = {
    "my_sample": {
        2: "NANANANANA",
        "one": "CATCANCAT",
        "X": "GATTACAGATTACAN"
    },
    "my_other_sample": {
        2: "GANGANGAN",
        "one": "TATANTATA",
        "X": "GATTACAGATTACAN"
    }
}
```

The sequences are stored in a nested structure of Python dictionaries, each key of the *sequence_data* dictionary represents the name or an otherwise unique identifier for a particular sample (e.g. "my_sample", "my_other_sample"),

the value is a dictionary whose own keys represent chromosome names or numbers [1] and the corresponding values are the sequences themselves as a string [2]. Regardless of how the chromosomes are identified, they must match across samples if one wishes to make comparisons across samples.

4.3 Providing Sequence Data as FASTA

If your sequences are in FASTA format, you must first index them with *samtools faidx*, then for each sample, pass the path to the index to Goldilocks in the following structure:

```
sequence_data = {
        "my_sequence": {"file": "/path/to/fastaidx/1.fa.fai"},
        "my_other_sequence": {"file": "/path/to/fastaidx/2.fa.fai"},
        "my_other_other_sequence": {"file": "/path/to/fastaidx/3.fa.fai"},
}
```

When supplying sequences in this format, note the following:

- *is_faidx=True* must be passed to the Goldilocks constructor (see below),
- It is assumed that the FASTA will be in the same directory with the same name as its index, just without the ".fai" extension,
- The key in the sequence data dictionary for each sample, must be *file*,
- The *i*-th sequence in each FASTA will be censused together, thus the order in which your sequences appear matters.

It is anticipated in future these assumptions will be circumvented by additional options to the Goldilocks constructor.

To specify the *is_faidx* argument, call the constructor like so:

```
g = Goldilocks(NucleotideCounterStrategy(["N"]), sequence_data, length=3, stride=1, is_faidx=True)
```

Now Goldilocks will know to expect to open these *file* values as FASTA indexes, not sequence data!

4.4 Conducting a Census

Once you have organised your sequence data in to the appropriate structure, you may conduct the census with Goldilocks by passing your strategy (e.g. NucleotideCounterStrategy) and sequence data to the imported `goldilocks.goldilocks.Goldilocks` class:

```
g = Goldilocks(NucleotideCounterStrategy(["N"]), sequence_data, length=3, stride=1)
```

Make sure you add the brackets after the name of the imported strategy, this 'creates' a usuable strategy for Goldilocks to work with.

For each chromosome (i.e. 'one', 'X' and 2) Goldilocks will split each sequence from the corresponding chromosome in each of the two example samples in to triplets of bases (as our specified region length is 3) with an offset of 1 (as our stride is 1). For example, chromosome *"one"* of *"my_sample"* will be split as follows:

```
CAT
 ATC
  TCA
   CAN
```

[1] Goldilocks has no preference for use of numbers or strings for chromosome names but it would be sensible to use numbers where possible for cases where you might wish to sort by chromosome.

[2] In future it is planned that sequences may be further nested in a dictionary to fully support polyploid species.

```
    ANC
     NCA
      CAT
```

In our example, the NucleotideCounterStrategy will then count the number of N bases that appear in each split, for each sample, for each chromosome.

4.5 Getting the Regions

Once the census is complete, you can extract all of the censused regions directly from your Goldilocks object. The example below demonstrates the format of the returned regions dictionary for the example data above:

```
> g.regions
{
    0: {
        'chr': 2,
        'ichr': 0,
        'pos_end': 3,
        'pos_start': 1,
        'group_counts': {
            'my_sample': {'default': 2},
            'my_other_sample': {'default': 1},
            'total': {'default': 3}
        },
    }

    ...

    27: {
        'chr': 'one',
        'ichr': 6,
        'pos_end': 9,
        'pos_start': 7,
        'group_counts': {
            'my_sample': {'default': 0},
            'my_other_sample': {'default': 0},
            'total': {'default': 0}
        },
    }
}
```

The returned structure is a dictionary whose keys represent the *id* of each region, with values corresponding to a dictionary of metadata for that particular *id*. The *id* is assigned incrementally (starting at 0) as each region is encountered by Goldilocks during the census and isn't particularly important.

Each region dictionary has the following metadata [3]:

Key	Value
id	A unique id assigned to the region by Goldilocks
chr	The chromosome the region appeared on (as found in the input data)
ichr	This region is the *ichr-th* to appear on this chromosome (0-indexed)
pos_start	The 1-indexed base of the sequence where the region begins (inclusive)
pos_end	The 1-indexed base of the sequence where the region ends (inclusive)

[3] Goldilocks used to feature a group_counts dictionary as part of the region metadata as shown in the example above, this was removed as it duplicated data stored in the group_counts variable in the Goldilocks object needlessly. It has not been removed in the example output above as it helps explain what regions represent.

In the example output above, the first (0th) censused region appears on chromosome 2 [4] and includes bases 1-3. It is the first (0th) region to appear on this chromosome and over those three bases, the corresponding subsequence for *"my_sample"* contained 2 N bases and the corresponding subsequence for *"my_other_sample"* contained 1. In total, over both samples, on chromosome 2, over bases 1-3, 3 N bases appeared.

The last region, region 27 (28th) appears on chromosome *"one"* [5] and includes bases 7-9. It is the seventh (6th by 0-index) found on this chromosome and over those three bases neither of the two samples contained an N base.

4.6 Sorting Regions

Following a census, Goldilocks allows you to sort the regions found by four mathematical operations: *max*, *min*, *mean* and *median*.

```
g_max = g.query("max")
g_min = g.query("min")
g_mean = g.query("mean")
g_median = g.query("median")
```

The result of a query is the original `goldilocks.goldilocks.Goldilocks` object with masked and sorted internal data. You can view a table-based representation of the regions with `goldilocks.goldilocks.Goldilocks.export_meta()`.

```
> g_max.export_meta(sep='\t', group="total")
[NOTE] Filtering values between 0.00 and 3.00 (inclusive)
[NOTE] 28 processed, 28 match search criteria, 0 excluded, 0 limit
chr     pos_start       pos_end total_default
2       1       3       3.0
2       3       5       3.0
2       5       7       3.0
2       7       9       3.0
2       2       4       2.0
2       4       6       2.0
2       6       8       2.0
2       8       10      2.0
X       13      15      2.0
one     4       6       2.0
one     5       7       2.0
one     3       5       1.0
one     6       8       1.0
X       1       3       0.0
X       2       4       0.0
X       3       5       0.0
X       4       6       0.0
X       5       7       0.0
X       6       8       0.0
X       7       9       0.0
X       8       10      0.0
X       9       11      0.0
X       10      12      0.0
X       11      13      0.0
X       12      14      0.0
one     1       3       0.0
one     2       4       0.0
one     7       9       0.0
```

[4] As numbers are ordered before strings like "one" and "X" in Python.
[5] As "X" is ordered before "one" in Python.

Note the regions in *g_max* are now sorted by the number of N bases that appeared. Ties are currently resolved by the region that was seen first (has the lowest *id*).

4.7 Setting Number of Processes

Goldilocks supports multiprocessing and can spawn some number of additional processes to perform the census steps before aggregating all the region counters and answering queries. To specify the number of processes Goldilocks should use, specify a *processes* argument to the constructor:

```
g = Goldilocks(NucleotideCounterStrategy(["N"]), sequence_data, length=3, stride=1, processes=4)
```

4.8 Full Example

Census an example sequence for appearance of 'N' bases:

```
from goldilocks import Goldilocks
from goldilocks.strategies import NucleotideCounterStrategy

sequence_data = {
    "my_sample": {
        2: "NANANANANA",
        "one": "CATCANCAT",
        "X": "GATTACAGATTACAN"
    },
    "my_other_sample": {
        2: "GANGANGAN",
        "one": "TATANTATA",
        "X": "GATTACAGATTACAN"
    }
}

g = Goldilocks(NucleotideCounterStrategy(["N"]), sequence_data, length=3, stride=1, processes=4)

g_max_n_bases = g.query("max")
g_min_n_bases = g.query("min")
g_median_n_bases = g.query("median")
g_mean_n_bases = g.query("mean")
```

CHAPTER 5

Advanced Package Usage

The following assumes basic Python programming experience (and that you have installed Goldilocks and familiarised yourself with the basics), skip to the end if you think you know what you're doing.

5.1 Filtering Regions

5.1.1 Group

By default when returning region data the "total" group is used, in our running example of counting missing nucleotides, this would represent the total number of 'N' bases seen in sequence data across each sample in the same genomic region on the same chromosome. But if you are more interested in a particular sample:

```
g.query("max", group="my_sample")
```

5.1.2 Track

When using tracks (for strategies that calculate multiple distinct values for each genomic region - such as different nucleotide bases or k-mers), you may wish to extract regions based on scores for a certain track:

```
g.query("max", track="AAA")
```

5.1.3 Absolute distance

You may be interested in regions within some distance of the mean:

```
g.query("mean", acutal_distance=10)
```

5.1.4 Percentile distance

Or perhaps the "top 10%", or the "middle 25%" around the mean:

```
g.query("max", percentile_distance=10)
g.query("mean", percentile_distance=25)
```

When not using *max* or *min*, by default both actual and percentile differences calculate 'around' the *mean* or *median* value instead. If you'd like to control this behaviour you can specify a direction: Let's fetch regions that have values falling within 25% above or below the mean respectively:

```
g.query("mean", percentile_distance=25, direction=1)
g.query("mean", percentile_distance=25, direction=-1)
```

5.1.5 Multiple criteria

You can of course use these at the same time (though actual and percentile distances are mutually exclusive), let's fetch the top 10% of regions that contain the most "AAA" k-mers for all chromosomes in a hypothetical sample called "my_sample":

```
g.query("max", group="my_sample", track="N", percentile_distance=10)
```

5.2 Excluding Regions

The filter function also allows users to specify a dictionary of exclusion criteria.

5.2.1 Starting position

To filter regions based on the 1-indexed starting position greater than or equal to 3:

```
g.query("min", exclusions={
                            "start_gte": 3,
                            })
```

5.2.2 Ending position

To filter regions based on the 1-indexed ending position less than or equal to 9:

```
g.query("min", exclusions={
                            "end_lte": 9,
                            })
```

5.2.3 Chromosome

You can filter regions that appear on particular chromosomes completely by providing a list:

```
g.query("min", exclusions={
                            "chr": ["X", 6],
                            })
```

5.2.4 Value of another count group

When using groups, one may wish to exclude results where the value of another group is less than the one selected by the query. For example, for each region the following would result in regions where the count for *my-other-sample* is greater than *my-sample*:

```
g.query("min", group="my-sample", exclusions={
                            "region_group_lte": "my-other-sample",
                            })
```

5.2.5 Multiple Criteria

You may want to use such exclusion criteria at the same time. Let's say we have a bunch of sequence data from a species whose chromosomes all feature centromeres between bases 500-1000. Let's ignore regions from that area. Let's also exclude anything from chromosome 'G'. If a single one of these criteria are true, a region will be excluded:

```
g.query("mean", exclusions={
                                "start_gte": 500,
                                "end_lte": 1000,
                                "chr": ['G'],
                                })
```

What if you want to exclude based on multiple criteria that should all be true? Let's exclude regions that start before or on base 100 on chromosome X or Y [1]. Note the use of *use_and=True*! [2]

```
g.query("mean", exclusions={
                                "start_lte": 100,
                                "chr": ['X', 'Y'],
                                }, use_and=True)
```

5.2.6 Chromosome specific criteria

Finally applying exclusions across all chromosomes might seem quite naive, what if we want to ignore centromeres on a real species? Introducing chromosome dependent exclusions; the syntax is the same as previously, just the exclusions dictionary is a dictionary of dictionaries with keys representing each chromosome. Note the use of *use_chrom=True*:

```
g.query("median", exclusions={
                                    "one": {
                                        "start_lte": 3,
                                        "end_gte": 4
                                    },
                                    2: {
                                        "start_gte": 9
                                    },
                                    "X": {
                                        "chr": True
                                    }}, use_chrom=True)
```

It is important to note that currently Goldilocks does not sanity check the contents of the exclusions dictionary including the spelling of exclusion names or whether you have correctly set use_chrom if you are providing chromosome specific filtering. However, on this latter point, if Goldilocks detects a key in the exclusions dictionary matches the name of a chromosome, it will print a warning (but continue regardless).

5.3 Limiting Regions

One may also limit the number of results returned by Goldilocks:

```
g.query("mean", limit=10)
```

[1] Support for chromosome matching is still 'or' based even when using use_and=True, a region can't appear on more than one chromosome and so this seemed a more natural and useful behaviour.

[2] Apart from the above caveat on chromosome matching always being or-based, currently there is no support for more complicated queries such as exclude if (statement1 and statement2) or statement3. It's or, or and on all criteria!

5.4 Full Example

Almost all of these options can be used together! Let's finish off our examples by finding the top 5 regions that are within an absolute distance of 1.0 from the maximum number of 'N' bases seen across all subsequences over the 'my_sample' sample. We'll exclude any region that appears on chromosome "one" and any regions on chromosome 2 that start on a base position greater than or equal to 5 *and* end on a base position less than or equal to 10. Although when filtering the default track is indeed 'default', we've explicity set that here too.:

```
g.query("max",
          group="my_sample",
          track="default",
          actual_distance=1,
          exclusions={
                2: {
                    "start_gte": 5,
                    "end_lte": 10
                },
                "one": {
                    "chr":True
                }
            },
            use_chrom=True,
            use_and=True,
            limit=5
).export_meta(sep="\t")

[NOTE] Filtering values between 1.00 and 2.00 (inclusive)
[NOTE] 28 processed, 12 match search criteria, 7 excluded, 5 limit
chr     pos_start       pos_end my_other_sample_default my_sample_default
2       1       3       1.0     2.0
2       3       5       1.0     2.0
2       2       4       1.0     1.0
2       4       6       1.0     1.0
X       13      15      1.0     1.0
```

CHAPTER 6

Exporting

Goldilocks provides functions for the exporting of all censused regions metadata or for filtered regions resulting from a query. The examples below follow on from the basic usage instructions earlier in the documentation.

6.1 Census Data

For a given sample one may export basic metadata for all regions that included sequence data from that particular sample. The header is as follows:

Key	Value
id	A unique id assigned to the region by Goldilocks
track1	The value for the region as calculated by the strategy used. By default if a list of tracks is not specified when the strategy is created, there will be just one track named 'default'. For the majority of 'basic' strategies this will be the case.
[track2 ... trackN]	Optional further fields will appear for additional tracks, the column header will feature the name of the track. For example, a k-mer counting strategy would feature a column for each k-mer specified to the strategy.
chr	The chromosome the region appeared on (as found in the input data)
pos_start	The 1-indexed base of the sequence where the region begins (inclusive)
pos_end	The 1-indexed base of the sequence where the region ends (inclusive)

Using the *my_sample* data:

```
...
g.export_meta("my_sample", sep="\t")

id      default chr     pos_start       pos_end
0       2       2       1       3
1       1       2       2       4
2       2       2       3       5
3       1       2       4       6
4       2       2       5       7
5       1       2       6       8
6       2       2       7       9
7       1       2       8       10
8       0       X       1       3
9       0       X       2       4
10      0       X       3       5
11      0       X       4       6
12      0       X       5       7
13      0       X       6       8
14      0       X       7       9
```

```
15      0       X       8       10
16      0       X       9       11
17      0       X       10      12
18      0       X       11      13
19      0       X       12      14
20      1       X       13      15
21      0       one     1       3
22      0       one     2       4
23      0       one     3       5
24      1       one     4       6
25      1       one     5       7
26      1       one     6       8
27      0       one     7       9
```

6.2 FASTA

From any sorting or filtering operation on censused regions, a new Goldilocks object is returned, providing function to output filtered sequence data to FASTA format.

Following on from the example introduced earlier, the example below shows the subsequences of *my_sample* in the FASTA format, ordered by their appearance in the filtered *candidates* list, from the highest number of 'N' bases, to the lowest.

```
...
candidates = g.query("max", group="my_sample")
candidates.export_fasta("my_sample")

>my_sample|Chr2|Pos1:3
NAN
>my_sample|Chr2|Pos3:5
NAN
>my_sample|Chr2|Pos5:7
NAN
>my_sample|Chr2|Pos7:9
NAN
>my_sample|Chr2|Pos2:4
ANA
>my_sample|Chr2|Pos4:6
ANA
>my_sample|Chr2|Pos6:8
ANA
>my_sample|Chr2|Pos8:10
ANA
>my_sample|ChrX|Pos13:15
CAN
>my_sample|Chrone|Pos4:6
CAN
>my_sample|Chrone|Pos5:7
ANC
>my_sample|Chrone|Pos6:8
NCA
>my_sample|ChrX|Pos1:3
GAT
>my_sample|ChrX|Pos2:4
ATT
>my_sample|ChrX|Pos3:5
TTA
```

```
>my_sample|ChrX|Pos4:6
TAC
>my_sample|ChrX|Pos5:7
ACA
>my_sample|ChrX|Pos6:8
CAG
>my_sample|ChrX|Pos7:9
AGA
>my_sample|ChrX|Pos8:10
GAT
>my_sample|ChrX|Pos9:11
ATT
>my_sample|ChrX|Pos10:12
TTA
>my_sample|ChrX|Pos11:13
TAC
>my_sample|ChrX|Pos12:14
ACA
>my_sample|Chrone|Pos1:3
CAT
>my_sample|Chrone|Pos2:4
ATC
>my_sample|Chrone|Pos3:5
TCA
>my_sample|Chrone|Pos7:9
CAT
```

CHAPTER 7

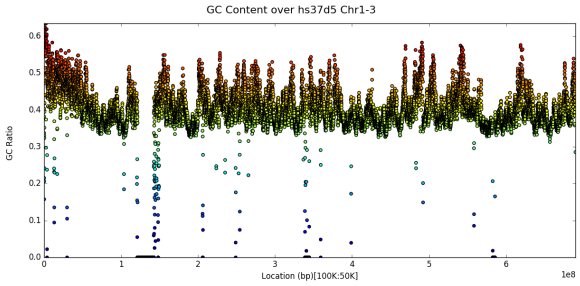
Plotting

7.1 Scatter Graphs

7.1.1 Simple Plot

After executing a census one can use the `plot` function to create a scatter graph of results. The `x` axis is the location along the genome (with ordered chromosomes or contigs appearing sequentially) and the `y` axis is the value of the censused region according to the strategy used. The example below plots GC content ratio across the first three chromosomes of the `hs37d5` reference sequence, with a window size of 100,000 and a step or overlap of 50,000. Note that the plot title may be specified with the `title` keyword argument.

```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy

sequence_data = {
    "my_sequence": {"file": "/store/ref/hs37d5.1-3.fa.fai"},
}
g = Goldilocks(GCRatioStrategy(), sequence_data, length="100K", stride="50K", is_faidx=True)
g.plot(title="GC Content over hs37d5 Chr1-3")
```

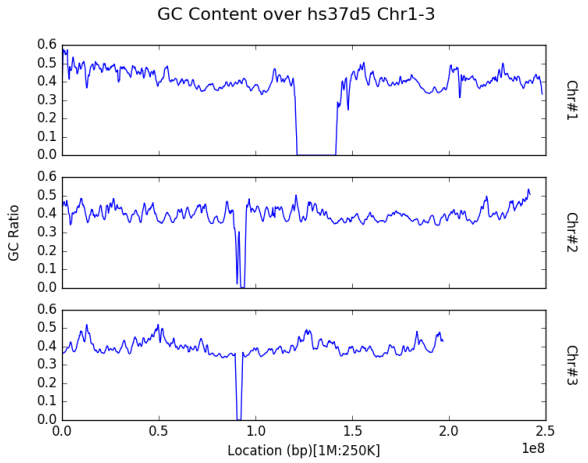


7.2 Line Graphs

7.2.1 Plot multiple contigs or chromosomes from one sample

For long genomes or a census with a small window size, simple plots as shown in the previous section can appear too crowded and thus difficult to extract information from. One can instead plot, for a given input sample, a panel of census region data, by chromosome by specifying the name of the sample as the first parameter to the `plot` function as per the example below:

```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy

sequence_data = {
    "hs37d5": {"file": "/store/ref/hs37d5.1-3.fa.fai"},
    "GRCh38": {"file": "/store/ref/Homo_sapiens.GRCh38.dna.chromosome.1-3.fa.fai"},
}
g = Goldilocks(GCRatioStrategy(), sequence_data, length="1M", stride="250K", is_faidx=True)
g.plot("hs37d5", title="GC Content over hs37d5 Chr1-3")
```

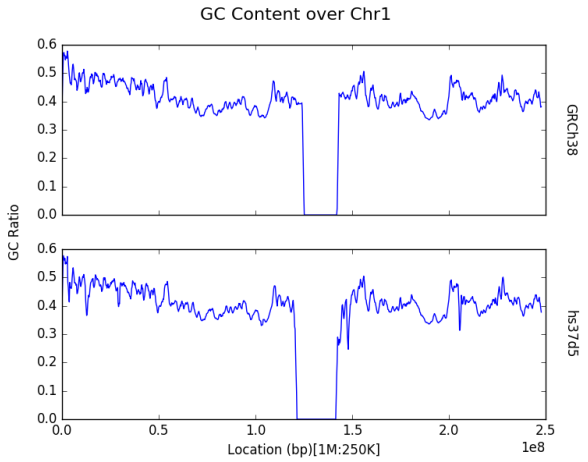


Note that both the `x` and `y` axes are shared between all panels to avoid the automatic creation of graphics with the potential to mislead readers on a first glance by not featuring the same axes ticks.

7.2.2 Plot a contig or chromosome from multiple samples

By default, data within the census is aggregated by region across all input samples (in the `sequence_data` dictionary) for the entire genome. However, one may be interested in comparisons across samples, rather than between chromosomes in a single sample. One can plot the census results for a specific contig or chromosome for each of the input samples, by specifying the `chrom` keyword argument to the `plot` function. Take note that the argument refers to the sequence that appears as the i'th contig of each of the input FASTA and not the actual name or identifier of the chromosome itself.

```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy

sequence_data = {
    "hs37d5": {"file": "/store/ref/hs37d5.1.fa.fai"},
    "GRCh38": {"file": "/store/ref/Homo_sapiens.GRCh38.dna.chromosome.1.fa.fai"},
}
g = Goldilocks(GCRatioStrategy(), sequence_data, length="1M", stride="250K", is_faidx=True)
g.plot(chrom=1, title="GC Content over Chr1")
```



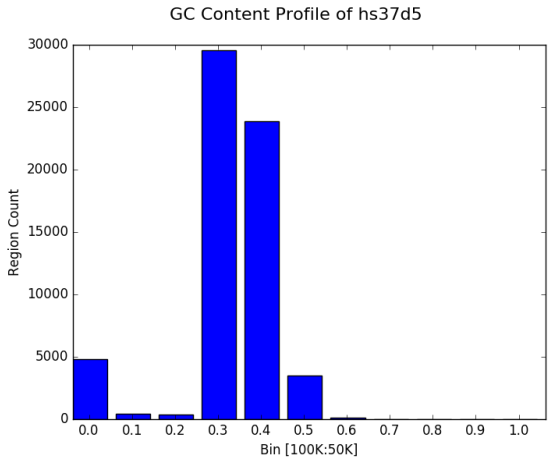
7.3 Histograms

7.3.1 Simple profile (binning) plot

Rather than inspection of individual data points, one may want to know how census data behaves as a whole. The `plot` function provides functionality to *profile* the results of a census through a histogram. Users can do this by providing a list of bins to the `bins` keyword argument of the `plot` function, following a census.

The example below shows the distribution of GC content ratio across the `hs37d5` reference sequence for all 100Kbp regions (and step of 50Kbp). The `x` axis is the bin and the `y` axis represents the number of censused regions that fell into a particular bin.

```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy

sequence_data = {
    "my_sequence": {"file": "/store/ref/hs37d5.fa.fai"}
}

g = Goldilocks(GCRatioStrategy(), sequence_data,
        length="100K", stride="50K", is_faidx=True)

g.plot(bins=[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
    title="GC Content Profile of hs37d5"
)
```



7.3.2 Simpler profile (binning) plot

It's trivial to select some sensible bins for the plotting of GC content as we know that the value for each region must fall between 0 and 1. However, many strategies will have an unknown minimum and maximum value and it can thus be difficult to select a suitable binning strategy without resorting to trial and error.

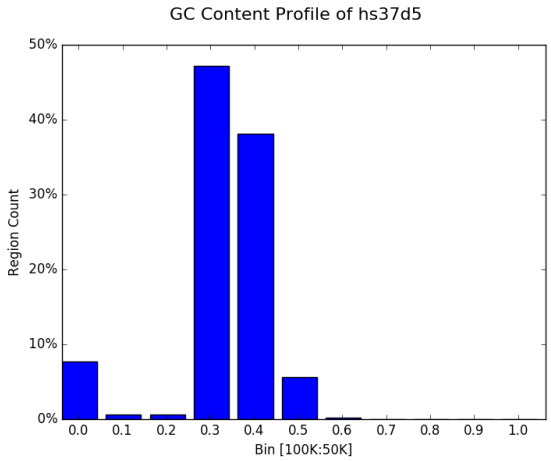
Thus the `plot` function permits a single integer to be provided to the `bins` keyword instead of a list. This will automatically create $N + 1$ equally sized bins (reserving a special bin for 0.0) between 0 and the maximum observed value for the census. It is also possible to manually set the size of the largest bin with the `bin_max` keyword argument. The following example creates the same graph as the previous subsection, but without explicitly providing a list of bins.

```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy

sequence_data = {
    "my_sequence": {"file": "/store/ref/hs37d5.fa.fai"},
}
g = Goldilocks(GCRatioStrategy(), sequence_data, length="100K", stride="50K", is_faidx=True)
g.plot(bins=10, bin_max=1.0, title="GC Content Profile of hs37d5")
```

7.3.3 Proportional bin plot

Often it can be useful to compare the size of bins in terms of their proportion rather than raw counts alone. This can be accomplished by specifying `prop=True` to `plot`. The `y` axis is now the percentage of all regions that were placed in a particular bin instead of the raw count.

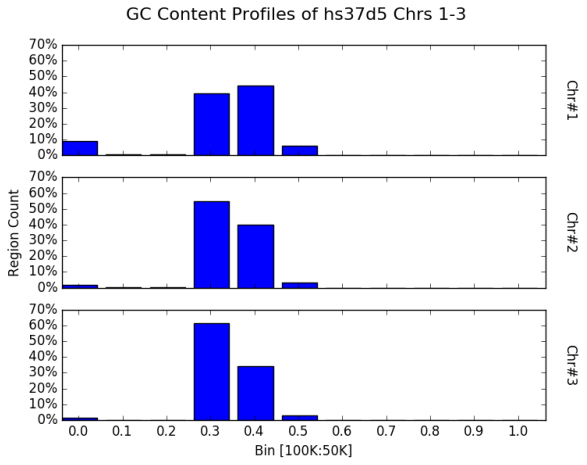
```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy

sequence_data = {
    "my_sequence": {"file": "/store/ref/hs37d5.fa.fai"}
}

g = Goldilocks(GCRatioStrategy(), sequence_data,
        length="100K", stride="50K", is_faidx=True)
g.plot(bins=10, bin_max=1.0, prop=True, title="GC Content Profile of hs37d5")
```



7.3.4 Bin multiple contigs or chromosomes from one sample

As demonstrated with the line plots earlier, one may also specify a sample name as the first parameter to `plot` to create a figure with each contig or chromosome's histogram on an individual panel.

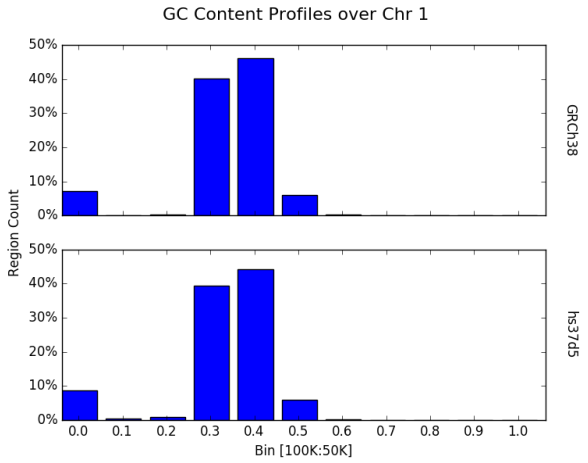
```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy

sequence_data = {
    "my_sequence": {"file": "/store/ref/hs37d5.1-3.fa.fai"}
}

g = Goldilocks(GCRatioStrategy(), sequence_data,
        length="100K", stride="50K", is_faidx=True)

g.plot("my_sequence",
    bins=10, bin_max=1.0, prop=True, title="GC Content Profiles of hs37d5 Chrs 1-3")
```



7.3.5 Bin a contig or chromosome from multiple samples

Similarly, one may want to profile a single contig or chromosome between each input group as previously demonstrated by the line graphs.

```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy

sequence_data = {
    "hs37d5": {"file": "/store/ref/hs37d5.1.fa.fai"},
    "GRCh38": {"file": "/store/ref/Homo_sapiens.GRCh38.dna.chromosome.1.fa.fai"}
}

g = Goldilocks(GCRatioStrategy(), sequence_data,
        length="100K", stride="50K", is_faidx=True)
g.plot(chrom=1, bins=10, bin_max=1.0, prop=True, title="GC Content Profiles over Chr 1")
```

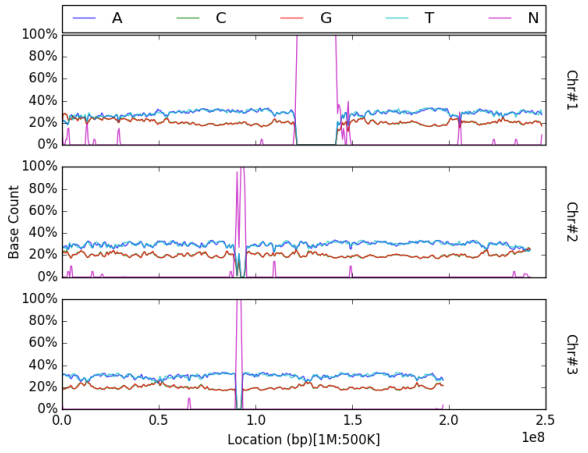


7.4 Advanced

7.4.1 Plot data from multiple counting tracks from one sample's chromosomes

The examples thus far have demonstrated plotting the results of a strategy responsible for counting one interesting property. But strategies are capable of counting multiple targets of interest simultaneously. Of course, one may wish to plot the results of all tracks rather than just the totals - especially for cases such as nucleotide counting where the sum of all counts will typically equal the size of the census region! The `plot` function accepts a list of track names to plot via the `tracks` keyword argument. Each counting track is then drawn on the same panel for the appropriate chromosome. A suitable legend is automatically placed at the top of the figure.

```
from goldilocks import Goldilocks
from goldilocks.strategies import NucleotideCounterStrategy

sequence_data = {
    "hs37d5": {"file": "/store/ref/hs37d5.1-3.fa.fai"},
}

g = Goldilocks(NucleotideCounterStrategy(["A", "C", "G", "T", "N"]), sequence_data,
        length="1M", stride="500K", is_faidx=True, processes=4)
g.plot(group="hs37d5", prop=True, tracks=["A", "C", "G", "T", "N"])
```

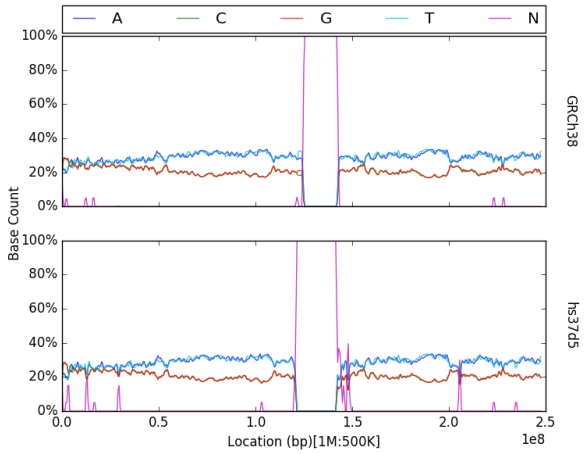


Note that `prop` is not a required argument, but can still be used with the `tracks` list to plot counts proportionally.

7.4.2 Plot data from multiple counting tracks for one chromosome across many samples

As previously demonstrated, one can use the `chrom` keyword argument for `plot` to create a figure featuring a panel per input sample, displaying census results for a particular chromosome. Similarly, this feature is supported when plotting multiple tracks with the `tracks` keyword.

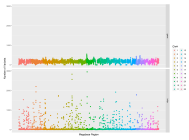
```
from goldilocks import Goldilocks
from goldilocks.strategies import NucleotideCounterStrategy

sequence_data = {
    "hs37d5": {"file": "/store/ref/hs37d5.1.fa.fai"},
    "GRCh38": {"file": "/store/ref/Homo_sapiens.GRCh38.dna.chromosome.1.fa.fai"},
}

g = Goldilocks(NucleotideCounterStrategy(["A", "C", "G", "T", "N"]), sequence_data,
        length="1M", stride="500K", is_faidx=True, processes=4)
g.plot(chrom=1, prop=True, tracks=["A", "C", "G", "T", "N"])
```



7.5 Integration with external plotting tools

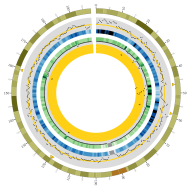
7.5.1 ggplot2

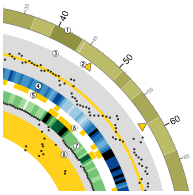
Plotting packages such as `ggplot2` favour ‘‘melted” input. The figure below was created using data from Goldilocks as part of our quality control study, the scatter plot compares the density of SNPs between the GWAS and SNP chip studies across the human genome.



7.5.2 Circos

Goldilocks has an output format specifically designed to output information for use with the ‘‘popular and pretty” `circos` visualisation tool. Below is an example of a figure that can be generated from data gathered by Goldilocks. The figure visualises the selection of regions from our original quality control study. The Python script used to generate the data and the Circos configuration follow.





Python script

```
from goldilocks import Goldilocks
from goldilocks.strategies import PositionCounterStrategy

sequence_data = {
    "gwas": {"file": "/encrypt/ngsqc/vcf/cd-seq.vcf.q"},
    "ichip": {"file": "/encrypt/ngsqc/vcf/cd-ichip.vcf.q"},
}

g = Goldilocks(PositionCounterStrategy(), sequence_data,
        length="1M", stride="500K", is_pos_file=True)

# Query for regions that meet all criteria across both sample groups
# The output file goldilocks.circ is used to plot the yellow triangular indicators
g.query("median", percentile_distance=20, group="gwas", exclusions={"chr": [6]})
g.query("max", percentile_distance=5, group="ichip")
g.export_meta(fmt="circos", group="total", value_bool=True, chr_prefix="hs", to="goldilocks.circ")

# Reset the regions selected and saved by queries
g.reset_candidates()
```

```
# Export all region counts for both groups individually
# The -all.circ files are used to plot the scatter plots and heatmaps
g.export_meta(fmt="circos", group="gwas", chr_prefix="hs", to="gwas-all.circ")
g.export_meta(fmt="circos", group="ichip", chr_prefix="hs", to="ichip-all.circ")

# Export region counts for the groups where the criteria are met
# The -candidates.circ files are used to plot the yellow 'bricks' that
#   appear between the two middle heatmaps
g.query("median", percentile_distance=20, group="gwas")
g.export_meta(fmt="circos", group="gwas", to="gwas-candidates.circ")
g.reset_candidates()
g.query("max", percentile_distance=5, group="ichip")
g.export_meta(fmt="circos", group="ichip", to="ichip-candidates.circ")
g.reset_candidates()
```

Circos configuration

```
# circos.conf
<colors>
gold = 255, 204, 0
</colors>

karyotype = data/karyotype/karyotype.human.hg19.txt
chromosomes_units           = 1000000
chromosomes_display_default = no
chromosomes = hs3;

<ideogram>

<spacing>
default = 0.01r
break   = 2u
</spacing>


# Ideogram position, fill and outline
radius    = 0.9r
thickness = 80p
fill      = yes
stroke_color     = dgrey
stroke_thickness = 3p

# Bands
show_bands = yes
band_transparency = 4
fill_bands            = yes
band_stroke_thickness = 2
band_stroke_color     = white

# Labels
show_label     = no
label_font       = default
label_radius     = 1r + 75p
label_size     = 72
label_parallel   = yes
label_case     = upper

</ideogram>
```

```
# Ticks
show_ticks          = yes
show_tick_labels    = yes

<ticks>

label_font       = default
radius               = dims(ideogram,radius_outer)
label_offset         = 5p
orientation          = out
label_multiplier     = 1e-6
color                = black
chromosomes_display_default = yes

<tick>
    spacing = 1u
    size = 10p
    thickness = 3p
    color = lgrey
    show_label = no
</tick>

<tick>
    spacing = 5u
    size = 20p
    thickness = 5p
    color = dgrey
    show_label = yes
    label_size = 24p
    label_offset = 0p
    format = %d
</tick>

<tick>
    spacing = 10u
    size = 30p
    thickness = 5p
    color = black
    show_label = yes
    label_size = 40p
    label_offset = 5p
    format = %d
</tick>

</ticks>

track_width = 0.05
track_pad   = 0.02
track_start = 0.95

<plots>
<plot>
    type            = scatter
    file        = goldilocks.circ
    r1      = 0.98r
    r0      = 0.95r
    orientation = out
```

```
    glyph = triangle
    #glyph_rotation = 180
    glyph_size = 50p

    color     = gold
    stroke_thickness = 2p
    stroke_color = black

    min = 0
    max = 1

</plot>
<plot>
    type  = scatter

    file    = gwas-all.circ
    r1      = 0.95r
    r0      = 0.80r

    fill = no
    fill_color       = black
    color = black_a1
    stroke_color     = black
    glyph            = circle
    glyph_size       = 12

    <backgrounds>
        <background>
            color = vlgrey
            y0    = 207
        </background>
        <background>
            color = vlgrey
            y1    = 207
            y0    = 179
        </background>
        <background>
            color = gold
            y1    = 179
            y0    = 148
        </background>
        <background>
            color = vlgrey
            y1    = 145
            y0    = 122
        </background>
        <background>
            color = vlgrey
            y1    = 122
            y0    = 0
        </background>
    </backgrounds>

    <axes>
        <axis>
            color     = white
            thickness = 1
            spacing   = 0.05r
```

```
        </axis>
    </axes>
    <rules>
        <rule>
            condition  = var(value) < 1
            show = no
        </rule>
    </rules>
</plot>
<plot>
    type    = heatmap
    file    = gwas-all.circ

    # color list
    color   = grey,vvlblue,vlblue,lblue,blue,dblue,vdblue,vvdblue,black
    r1      = 0.80r
    r0      = 0.75r

    scale_log_base = 0.75
    color_mapping = 2
    min = 1
    max = 267 # 95%
</plot>

<plot>
    type            = tile
    layers_overflow = collapse
    file        = gwas-candidates.circ
    r1      = 0.7495r
    r0      = 0.73r
    orientation = in

    layers      = 1
    margin      = 0.0u
    thickness   = 30p
    padding     = 8p

    color     = gold
    stroke_thickness = 0
    stroke_color = gold
</plot>
<plot>
    type            = tile
    layers_overflow = collapse
    file        = ichip-candidates.circ
    r1      = 0.73r
    r0      = 0.70r
    orientation = out

    layers      = 1
    margin      = 0.0u
    thickness   = 30p
    padding     = 8p

    color     = gold
    stroke_color = gold
</plot>
```

```
<plot>
    type    = heatmap
    file    = ichip-all.circ

    # color list
    color   = grey,vvlgreen,vlgreen,lgreen,green,dgreen,vdgreen,vvdgreen,black
    r1      = 0.70r
    r0      = 0.65r

    min = 1
    max = 1097.71 # 99%
    color_mapping = 2
    scale_log_base = 0.2

</plot>
<plot>
    type  = scatter

    file    = ichip-all.circ
    r1      = 0.65r
    r0      = 0.50r
    orientation = in

    fill_color       = black
    stroke_color     = black
    glyph            = circle
    glyph_size       = 12
    color = black_a1

    <backgrounds>
        <background>
            color = gold
            y0    = 379
        </background>
        <background>
            color = vlgrey
            y1    = 379
            y0    = 49
        </background>
        <background>
            color = vlgrey
            y1    = 49
            y0    = 0
        </background>
    </backgrounds>

    <axes>
        <axis>
            color     = white
            thickness = 1
            spacing   = 0.05r
        </axis>
    </axes>
    <rules>
        <rule>
            condition  = var(value) < 1
            show = no
        </rule>
```

```
    </rules>
</plot>

</plots>

#################################################################
# The remaining content is standard and required. It is imported
# from default files in the Circos distribution.
#
# These should be present in every Circos configuration file and
# overridden as required. To see the content of these files,
# look in etc/ in the Circos distribution.

<image>
# Included from Circos distribution.
<<include etc/image.conf>>
</image>

# RGB/HSV color definitions, color lists, location of fonts, fill patterns.
# Included from Circos distribution.
<<include etc/colors_fonts_patterns.conf>>

# Debugging, I/O an dother system parameters
# Included from Circos distribution.
<<include etc/housekeeping.conf>>
anti_aliasing* = no
```

CHAPTER 8

Custom Strategies

One of the major features of Goldilocks is its extensibility. Strategies are both easily customisable and interchangeable, as they all share a common interface. This interface also provides a platform for users with some knowledge of Python to construct their own custom census rules. One such example follows below:

8.1 A Simple ORF Finder

8.1.1 Code Sample

```
# Import Goldilocks and the BaseStrategy class
from goldilocks import Goldilocks
from goldilocks.strategies import BaseStrategy

# Define a new class for your custom strategy that inherits from BaseStrategy
class MyCustomSimpleORFCounterStrategy(BaseStrategy):

    # Initialising function boilerplate, required to set-up some properties of the census
    def __init__(self, tracks=None, min_codons=1):
        # Initialise the custom class with super
        super(MyCustomSimpleORFCounterStrategy, self).__init__(
                tracks=range(0,3),                  # Use range to specify a counter for
                                                    #  each of the three possible forward
                                                    #  reading frames in which to search
                                                    #  to search for open reading frames
                label="Forward Open Reading Frames"  # Y-Axis Plot Label
        )
        self.MIN_CODONS = min_codons

    # This function defines the actual behaviour of a census for a given region
    #  of sequence and the current counting track (one of three reading frames)
    def census(self, sequence, track_frame, **kwargs):
        STARTS = ["ATG"]
        STOPS = ["TAA", "TGA", "TAG"]
        CODON_SIZE = 3

        # Split input sequence into codons. Open a frame if a START is found
        #  and increment the ORF counter if a STOP is encountered afterward
        orfs = orf_open = 0
        for i in xrange(track_frame, len(sequence), CODON_SIZE):
            codon = sequence[i:i+CODON_SIZE].upper()
            if codon in STARTS and orf_open == 0:
```

```
                orf_open = 1
            elif codon in STOPS and orf_open > 0:
                if orf_open > self.MIN_CODONS:
                    orfs += 1
                orf_open = 0
            elif orf_open > 0:
                orf_open += 1
        return orfs

# Organise and execute the census
sequence_data = { "hs37d5": {"file": "/store/ref/hs37d5.1-3.fa.fai"} }
g = Goldilocks(MyCustomSimpleORFCounterStrategy(min_codons=30), sequence_data,
        length="1M", stride="1M", is_faidx=True, processes=4)
```

8.1.2 Implementation Description

Strategies are defined as Python classes, inheriting from the `BaseStrategy` class found in the `goldilocks.strategies` subpackage. The class requires just two function definitions to be compliant with the shared interface; `__init__`: the class initializer that takes care of the setup of the strategy's internals via the `BaseStrategy` parent class, and `census`: the function actually responsible for the behaviour of the strategy itself.

The example presented is a very simple open reading frame counter. It searches the three forward frames for start codons that are then followed by one of the three stop codons. The ''tracks" in this example are the three possible frames. Note on line 9 that our `__init__` provides a default argument for `tracks` of `None`. Thus this particular strategy does not need the `tracks` argument. Instead, the track list is provided by the strategy itself, and passed to the `BaseStrategy` `__init__` (line 12), forcing tracks to be the list [0, 1, 2]. The elements of this list are used as an integer offset from which to begin splitting input DNA sequences when conducting the census later, which is why on this occasion we don't want to allow the user to specify their own tracks. Other strategies, such as the included `NucleotideCounterStrategy` just pass the `tracks` argument from the user through to the super `__init__`.

For a given array of `sequence` data and a frame offset (`track_frame`), the `census` function splits the sequence into nucleotide triplets from the offset and searches for open reading frames. A subsequence is considered an ORF by this strategy if the ATG START codon is encountered and later followed by any STOP codon.

Our example finishes with the familiar specification of the location of input sequence data and the construction of the census itself. Here we specify a census of all 1Mbp regions with no overlap (that is, the stride is equal to the size of the regions) and instantiate our new `MyCustomSimpleORFCounterStrategy` with a keyword requiring valid ORFs to be at least 30 codons in length (excluding start and stop).

Every strategy's `census` function is expected to return a numerical result that can be used to rank and sort regions, in this scenario, `census` returns the number of ORFs found.

Note also, strategies may specify any number of keyword arguments that are not found in the `BaseStrategy`. In our example, `min_codons` can be set by a user to specify how many codons must lie between an opening and closing codon to be counted as an open reading frame. We store this value as a member of the strategy object on line 18 and use it on line 35 to ensure the `orfs` counter is only incremented when the length of the current open reading frame has exceeded the provided threshold. One could store any number of configurable parameters inside of the strategy class in this fashion. This framework allows one to increase the complexity of strategies while still providing a friendly and interchangeable interface for end users.

CHAPTER 9

Examples

The following includes some simple examples of what Goldilocks can be used for.

9.1 Example One

Read a pair of 1-indexed base position lists and output all regions falling within 2 of the maximum count of positions in regions across both, in a table.

```
from goldilocks import Goldilocks
from goldilocks.strategies import PositionCounterStrategy
data = {
        "my_positions": {
            1: [1,2,5,10,15,15,18,25,30,50,51,52,53,54,55,100]
        },
        "my_other_positions": {
            1: [1,3,5,7,9,12,15,21,25,51,53,59,91,92,93,95,99,100]
        }
}
g = Goldilocks(PositionCounterStrategy(), data, is_pos=True, length=10, stride=1)

g.query("max", actual_distance=2).export_meta(sep="\t", group="total")
```

9.2 Example Two

Read a short sequence, census GC-ratio and output the top 5 regions as FASTA.

```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy
data = {
        "my_sequence": {
            1: "ACCGAGAGATTT"
        }
}
g = Goldilocks(GCRatioStrategy(), data, 3, 1)

g.query("max", limit=5).export_fasta()
```

9.3 Example Three

Read a short sequence and census the appearance of the "AAA" and "CCC" motif. Output a table of regions with the most occurrences of CCC (and at least one) and another table of regions featuring the most appearances of both motifs. Output only the maximum region (actual_distance = 0) displaying both motifs to FASTA.

```
from goldilocks import Goldilocks
from goldilocks.strategies import MotifCounterStrategy
data = {
        "my_sequence": {
            1: "CCCAAACCCGGGCCCGGGAGAAACCC"
        }
}
g = Goldilocks(MotifCounterStrategy(["AAA", "CCC"]), data, 9, 1)

g.query("max", track="CCC", gmin=1).export_meta(sep="\t")
g.query("max", group="total").export_meta(sep="\t", group="total", track="default")

g.query("max", group="total", actual_distance=0).export_fasta()
```

9.4 Example Four

Read two samples of three short chromosomes and search for 'N' nucleotides. List and export a FASTA of regions that contain at least one N, sorted by number of N's appearing across both samples. Below, an example of complex filtering.

```
from goldilocks import Goldilocks
from goldilocks.strategies import NucleotideCounterStrategy
data = {
    "sample_one": {
        1: "ANAGGGANACAN",
        2: "ANAGGGANACAN",
        3: "ANANNNANACAN",
        4: "NNNNAANNAANN"
    },
    "sample_two": {
        1: "ANAGGGANACAN",
        2: "ANAGGGANACAN",
        3: "ANANNNANACAN",
        4: "NNNANNAANNAA"
    }
}
g = Goldilocks(NucleotideCounterStrategy(["N"]), data, 3, 1)

g_max = g.query("max", gmin=1)
g_max.export_meta(sep="\t")
g_max.export_fasta()

g.query("min",
        gmin = 1,
        exclusions={
            # Filter any region with a starting position <= 3 or >= 10
            "start_lte": 3,
            "start_gte": 10,
```

```
            # Filter any regions on Chr1
            1: {
                "chr": True
            },

            # Filter NO regions on Chr2
            # NOTE: This also prevents the superexclusions above being applied.
            2: {
                "chr": False
            },

            # Filter any region on Chr3 with an ending postion >= 9
            3: {
                "start_lte": 5 # NOTE: This overrides the start_lte applied above
            }
        }, use_chrom=True).export_meta(sep="\t")
```

9.5 Example Five

Read in four simple chromosomes from one sample and census the GC ratio. Plot both a scatter plot of all censused regions over both of the provided samples with position over the x-axis and value on the y-axis. Produce a second plot drawing a panel with a line graph for each chromosome with the same axes but data from one sample only. For the combined result of both samples and chromosomes, organise the result of the census for each region into desired bins and plot the result as a histogram. Repeat the process for the my_sequence sample and produce a panelled histogram for each chromosome.

```
from goldilocks import Goldilocks
from goldilocks.strategies import GCRatioStrategy
data = {
    "my_sequence": {
        1: "ANAGGGANACANANAGGGANACANANAGGGANACANANAGGGANACANANAGGGACGCGCGCGGGGANACAN"*500,
        2: "ANAGGCGCGCNANAGGGANACGCGGGGCCCGACANANAGGGANACANANAGGGACGCGCGCGCGCCCGACAN"*500,
        3: "ANAGGCGCGCNANAGGGANACGCGGGGCCCGACANANAGGGANACANANAGGGACGCGCGCGCGCCCGACAN"*500,
        4: "GCGCGCGCGCGCGCGCGGGGGGGGGGCGCCGCCNNNNNNNNNNNNNNNNNGCGCGCGCGCGCGCGNNNNNNNNN"*500
    },
    "my_same_sequence": {
        1: "ANAGGGANACANANAGGGANACANANAGGGANACANANAGGGANACANANAGGGACGCGCGCGGGGANACAN"*500,
        2: "ANAGGCGCGCNANAGGGANACGCGGGGCCCGACANANAGGGANACANANAGGGACGCGCGCGCGCCCGACAN"*500,
        3: "ANAGGCGCGCNANAGGGANACGCGGGGCCCGACANANAGGGANACANANAGGGACGCGCGCGCGCCCGACAN"*500,
        4: "GCGCGCGCGCGCGCGCGGGGGGGGGGCGCCGCCNNNNNNNNNNNNNNNNNGCGCGCGCGCGCGCGNNNNNNNNN"*500
    }
}
g = Goldilocks(GCRatioStrategy(), data, 50, 10)

g.plot()
g.plot("my_sequence")
g.profile(bins=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
g.profile("my_sequence", bins=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
```

9.6 Example Six

Read a set of simple chromosomes from two samples and tabulate the top 10% of regions demonstrating the worst consensus to the given reference over both samples. Plot the lack of consensus as line graphs for each chromosome,

for each sample, then over all chromosomes for all samples on one graph.

```
from goldilocks import Goldilocks
from goldilocks.strategies import ReferenceConsensusStrategy
data = {
    "first_sample": {
        1: "NNNAANNNNNCCCCCNNNNNGGGGGNNNNNTTTTTNNNNNAAAAANNNNNCCCCCNNNNNGGGGGNNNNNTTTTTNNNNN",
        2: "NNNNNCCCCCNNNNNTTTTTNNNNNAAAAANNNNNGGGGGNNNNNCCCCCNNNNNTTTTTNNNNNAAAAANNNNNGGGGN"
    },
    "second_sample": {
        1: "NNNNNNNNNCCCCCCCCCCNNNNNNNNNTTTTTTTTTTNNNNNNNNNCCCCCCCCCCNNNNNNNNNTTTTTTTTT",
        2: "NNCCCCCCCCNNNNNNNNNAAAAAAAAAANNNNNNNNNCCCCCCCCCCNNNNNNNNNAAAAAAAAAANNNNNNNNN"
    }
}
ref = {
    1: "AAAAAAAAACCCCCCCCCGGGGGGGGGTTTTTTTTTAAAAAAAAACCCCCCCCCGGGGGGGGGTTTTTTTTT",
    2: "CCCCCCCCCTTTTTTTTTAAAAAAAAAGGGGGGGGGCCCCCCCCCTTTTTTTTTAAAAAAAAAGGGGGGGGG"
}

g = Goldilocks(ReferenceConsensusStrategy(reference=ref, polarity=-1), data, stride=10, length=10)
g.query("max", percentile_distance=10).export_meta(group="total", track="default")

g.plot("first_sample")
g.plot("second_sample")
g.plot()
```

9.7 Example Seven

Read a pair of 1-indexed base position lists from two samples. Sort regions with the least number of marked positions on Sample 1 and subsort by max marked positions in Sample 2.

```
from goldilocks import Goldilocks
from goldilocks.strategies import PositionCounterStrategy
data = {
        "my_positions": {
            1: [1,2,3,4,5,6,7,8,9,10,
                11,13,15,17,19,
                21,
                31,39,
                41]
        },
        "other_positions": {
            1: [21,22,23,24,25,26,27,28,
                31,33,39,
                41,42,43,44,45,46,47,48,49,50]
        }
}
g = Goldilocks(PositionCounterStrategy(), data, is_pos=True, length=10, stride=5)

g.query("max", group="my_positions").query("max", group="other_positions").export_meta(sep="\t")
```

CHAPTER 10

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:

10.1 Types of Contributions

10.1.1 Report Bugs

Report bugs at https://github.com/samstudio8/goldilocks/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.

10.1.2 Fix Bugs

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

10.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

10.1.4 Write Documentation

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

10.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/samstudio8/goldilocks/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 :)

10.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/goldilocks.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 goldilocks
$ cd goldilocks/
$ 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 goldilocks tests
$ python setup.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.

10.3 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 2.6, 2.7, and 3.3, 3.4, and for PyPy. Check https://travis-ci.org/samstudio8/goldilocks/pull_requests and make sure that the tests pass for all supported Python versions.

10.4 Tips

To run a subset of tests:

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

CHAPTER 11

Credits

11.1 Development Lead

- Sam Nicholls <sam@samnicholls.net>

11.2 Contributors

None yet. Why not be the first?

CHAPTER 12

History

12.1 0.1.1 (2016-07-07)

- **Updated citation.** Please cite us! <3
- **[PR:ar0ch] Add lowercase matching in GCRatioStrategy** Fixes 'feature' where lowercase letters are ignored by GCRatioStrategy.

12.2 0.1.0 (2016-03-08)

- Goldilocks is published software!

12.3 0.0.83-beta

- *-l* and *-s* CLI arguments and corresponding *length* and *stride* parameters to *Goldilocks* constructor now support SI suffixes: *K*, *M*, *G*, *T*. *util* module contains *parse_si_bp* used to parse option strings and return the number of bases for length and stride.
- Add length and stride to x-axis label of plots.
- Add *ignore_query* option to *plot* to override new default behaviour of plot that only plots points for regions remaining after a call to *query*.
- Remove *profile* function, use *plot* with *bins=N* instead.
- Add binning to *plot* to reduce code duplication.
- Add *chrom* kwarg to *plot* to allow plotting of a single chromosome across multiple input genomes.
- Fix support for plotting data from multiple contigs or chromosomes of a single input genome when provided as a FASTA.
- Add *ignore_query* kwarg to *plot* for ignoring the results of a query on the *Goldilocks* object when performing a plot afterwards.
- Bins no longer have to be specified manually, use *bins=N*, this will create N+1 bins (a special 0 bin is reserved) between 0 and the largest observed value unless *bin_max* is also provided.
- Bins may have a hard upper limit set with *bin_max*. This will override the default of the largest observed value regardless of whether *bin_max* is smaller.
- Plots can now be plotted proportionally with *prop=True*.

- Improve labels for plotting.
- Reduce duplication of plotting code inside *plot*.
- Share Y axis across plot panels to prevent potentially misleading default plots.
- Reduce duplication of code used for outputting metadata:
- **Add *fmt* kwarg to *export_meta* that permits one of:**
  - **bed** BED format (compulsory fields only)
  - **circos** A format compatible with the circos plotting tool
  - **melt** A format that will suit import to an R dataframe without the need for additional munging with reshape2
  - **table** A plain tabular format that will suit for quick outputs with some munging
- Remove *print_melt*, use *export_meta* with *fmt=melt*.
- Add *is_pos_file* kwarg to Goldilocks, allows user to specify position based variants in the format *CHRtPOS* or *CHR:POS* in a newline delimited file.
- Changed required *idx* key to *file* in sequence dictionaries.
- Added custom strategy and plotting examples to the documentation.
- The *Goldilocks* class is now imported as *from goldilocks import Goldilocks*.
- The *textwrap.wrap* function is used to write out FASTA more cleanly.
- A serious regression in the parsing of FASTA files introduced by v0.0.80 has been closed.
- Improved plotting functionality for co-plotting groups, tracks of chromosome has been introduced. Tracks can now be plotted together on the same panel by providing their names as a list to the *tracks* keyword.
- *reset_candidates* allows users to "reset" the Goldilocks object after a query or sort has been performed on the regions.

12.4 0.0.82 (2016-01-29)

- Changed example to use *MotifCounterStrategy* over removed *KMerCounterStrategy*.
- Fix runtime *NameError* preventing *PositionCounterStrategy* from executing correctly.
- Fix runtime *NameError* preventing *ReferenceConsensusStrategy* from executing correctly.
- Add default *count* track to *PositionCounterStrategy* to prevent accidental multiple counting issue encountered when couting with the *default* track.
- Add LICENSE
- Paper accepted for press!

12.5 0.0.81 (2016-01-29)

- Fix versioning error.

12.6 0.0.80 (2015-08-10)

- Added multiprocessing capabilities during census step.
- Added a simple command line interface.
- Removed prepare-evaluate paradigm from strategies and now perform counts directly on input data in one step.
- Skip slides (and set all counts to 0) if their *end_pos* falls outside of the region on that particular genome's chromosome/contig.
- Rename *KMerCounterStrategy* to *MotifCounterStrategy*
- Fixed bug causing *use_and* to not work as expected for chromosomes not explicitly listed in the *exceptions* dict when also using *use_chrom*.
- Support use of FASTA files which must be supplied with a *samtools faidx* style index.
- Stopped supporting Python 3 due to incompatability with *buffer* and *memoryview*.
- Prevent *query* from deep copying itself on return. Note this means that a query will alter the original Goldilocks object.
- Now using a 3D numpy matrix to store counters with memory shared to support multiprocessing during census.
- Removed *StrategyValue* as these cannot be stored in shared memory. This makes ratio-based strategies a bit of a hack currently (but still work...)
- tldr; Goldilocks is at least 2-4x faster than previously, even without multiprocessing

12.7 0.0.71 (2015-07-11)

- Officially add MIT license to repository.
- Deprecate *_filter*.
- Update and tidy *examples.py*.
- *is_seq* argument to initialisation removed and replaced with *is_pos*.
- Use *is_pos* to indicate the expected input is positional, not sequence.
- Force use of *PositionCounterStrategy* when *is_pos* is True.
- **Sequence data now read in to 0-indexed arrays to avoid the overhead of string** re-allocation by having to append a padding character to the beginning of very long strings.
- Region metadata continues to use 1-indexed positions for user output.
- *VariantCounterStrategy* now *PositionCounterStrategy*.
- ***PositionCounterStrategy* expects 1-indexed lists of positions;** *prepare* populates the listed locations with 1 and then *evaluate* returns the sum as before.
- ***test_regression2* updated to account for converting 1-index to 0-index when** manually handling the sequence for expected results.
- *query* accepts *gmax* and *gmin* arguments to filter candidate regions by the group-track value.
- *CandidateList* removed and replaced with simply returning a new *Goldilocks*.

12.8 0.0.6 (2015-06-23)

- *Goldilocks.sorted_regions* stores a list of region ids to represent the result of a sorting operation following a call to *query*.
- Regions in *Goldilocks.regions* now always have a copy of their "id" as a key.
- *__check_exclusions* now accepts a *group* and *track* for more complex exclusion-based operations.
- *region_group_lte* and *region_group_gte* added to usable exclusion fields to remove regions where the value of the desired group/track combination is less/greater than or equal to the value of the group/track set by the current *query*.
- *query* now returns a new *Goldilocks* instance, rather than a *CandidateList*.
- *Goldilocks.candidates* property now allows access to regions, this property will maintain the order of *sorted_regions* if it has one.
- *export_meta* now allows *group=None*
- *CandidateList* class deleted.
- Test data that is no longer used has been deleted.
- Scripts for generating test data added to *test_gen/* directory.
- Tests updated to reflect the fact *CandidateList* lists are no longer returned by *query*.
- *_filter* is to be deprecated in favour of *query* by 0.0.7

12.9 Beta (2014-10-08)

- Massively updated! Compatability with previous versions very broken.
- Software retrofitted to be much more flexible to support a wider range of problems.

12.10 0.0.2 (2014-08-18)

- Remove incompatible use of *print*

12.11 0.0.1 (2014-08-18)

- Initial package

CHAPTER 13

Indices and tables

- genindex
- modindex
- search