SLiMBench Manual

Richard J. Edwards © 2012.

Contents

Tables

1. Introduction

This manual gives an overview of SLiMBench, a utility for generating and assessing the benchmarking of *de novo* SLiM discovery methods. Originally designed for the benchmarking of QSLiMFinder, the intention is to make something that can work on a number of SLiM discovery tools if the output is reformatted accordingly. (Format parsers will be added with time.)

SLiMBench essentially has two primary functions, which can be run independently:

1. Generating SLiM prediction benchmarking datasets from ELM (or other data in a similar format). This includes options for generating random and/or simulated datasets for ROC analysis etc.

2. Assessing the results of SLiM predictions against a Benchmark. This program is designed to work with SLiMFinder and QSLiMFinder output, so some prior results parsing may be needed for other methods.

The fundamentals are covered in Chapter 2, Fundamentals, including input and output details. Later sections give more details on how the methods work and statistics are generated. General details about Command-line options can be found in the RJE Python Appendices document included with this download. Details of command-line options specific to SLiMBench can be found in the distributed readme.html file and online at bioware.soton.ac.uk.

Like the software itself, this manual is a 'work in progress' to some degree. If the version you are now reading does not make sense, then it may be worth checking the website to see if a more recent version is available, as indicated by the Version section of the manual. Options may have been added over the past few weeks and not all found their way into the manual yet. Check the readme on the website for up-to-date options *etc*. In particular, default values for options are subject to change and should be checked in the readme.

Good luck!

Rich Edwards, 2012.

1.1. Version

This manual is designed to accompany SLiMBench version 1.7.

The manual was last edited on 21 February 2013.

1.2. Copyright, License and Warranty

SLiMBench is Copyright © 2012 Richard J. Edwards.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

The GNU General Public License should have been supplied with the SLiMBench program and is also available at www.gnu.org.

1.3. Using this Manual

As much as possible, I shall try to make a clear distinction between explanatory text (this) and text to be typed at the command-prompt etc. Command prompt text will be `written in Courier New` to make the distinction clearer. Program options, also called 'command-line parameters', will be **`written in bold Courier New`** (and coloured **`red`** for fixed portions or **`dark`** for user-defined portions, such as file names etc.). Command-line examples will be given in (grey) *`italicised Courier New`*. Optional parameters will (if I remember) be `[`in square brackets`]`. Names of files will be marked in coloured normal text.

1.4. Why use SLiMBench?

Benchmarking of SLiM discovery currently suffers from a few problems. First, there is no stable "gold standard" against which to benchmark. Although the ELM database provides such data, and is frequently used, the rules/decisions used to generate a good benchmark from ELM are not always clear: ELM does not exist in order to provide a SLiM discovery benchmark and thus its entries are not optimised, or always appropriate, for this task. Second, there is no formal definition of a successful prediction. SLiM discovery results are usually compared manually to know motifs and a rather subjective decision is made regarding whether there is a match. Third, the interdependency of SLiMs and the proteins in which they are found means that SLiM discovery methods will frequently return a genuine motif that was not the focus of the creation of the dataset.

SLiMBench aims to combat these problems. First, it provides a structured and documented pipeline for constructing True-Positive-containing datasets that are geared specifically towards SLiM discovery. This will not only make assessment of methods fairer – there is no point in testing a method against a dataset for which it has no chance of returning a motif because it is, for example, too degenerate or has too few occurrences – but it will also make assembly of newer and larger benchmarking datasets easier as databases such as ELM continue to expand. Second, by controlling the benchmarking data, SLiMBench can impose strict rules based on CompariMotif matches that score whether a returned motif is a hit to a known motif (either a True Positive or Off-target). Third, the statistics returned include different ways of handling the genuine "Off-target" motifs that are returned be methods – either classing them as True Positives, False Positives, or just ignoring them. This allows the true Precision of the method to be framed with upper and lower bounds.

1.5. Getting Help

Much of the information here is also contained in the documentation of the Python modules themselves. A full list of command-line parameters can be printed to screen using the **help** option, with short descriptions for each one:

```
python slimbench.py help
```

General details about Command-line options can be found in the RJE Python Appendices document included with this download. Details of command-line options specific to SLiMBench can be found in the distributed readme.html file and online at bioware.soton.ac.uk.

If still stuck, then please e-mail me (seqsuite@gmail.com) whatever question you have. If it is the results of an error message, then please send me that and/or the log file (see Chapter 2) too.

1.5.1. Something Missing?

As much as possible, the important parts of the software are described in detail in this manual. If something is not covered, it is generally not very important and/or still under development, and can therefore be safely ignored. If, however, curiosity gets the better of you, and/or you think that something important is missing (or badly explained), please contact me.

1.6. Citing SLiMBench

Until published, please cite the Seqsuite Website: https://sites.google.com/site/seqsuite/.

1.7. Availability and Local Installation

SLiMBench is distributed as a number of open source Python modules as part of the SeqSuite package. It should work on any system with Python installed without any extra setup required. If you do not have Python, you can download it free from www.python.org at http://www.python.org/download/. The modules are written in Python 2.7. The Python website has good information about how to download and install Python but if you have any problems, please get in touch and I will help if I can.

All the required files should have been provided in the downloaded zip file. Details can be found at http://bioware.soton.ac.uk/ and the accompanying RJE Python Appendices document. The Python Modules are open source and may be changed if desired, although please give me credit for any useful bits you pillage. I cannot accept any responsibility if you make changes and the program stops working, however!

Note that the organisation of the modules and the complexity of some of the classes is due to the fact that most of them are designed to be used in a number of different tools. As a result, not all the options listed in the __doc__() (`help`) will be of relevance. If you want some help understanding the way the modules and classes are set up so you can edit them, just contact me.

1.7.1. Programs Used by SLiMBench

In addition to the python modules listed above, SLiMBench makes use of the following published programs. These are freely available for downloading and installing. It is recommended that the user downloads and installs these programs according to the instructions given on the appropriate website.

BLAST: BLAST (Altschul et al. 1990) is freely available for download from NCBI at: http://www.ncbi.nlm.nih.gov/blast/download.shtml.

2. Fundamentals

2.1. Running SLiMBench

2.1.1. The Basics

If you have python installed on your system, you should be able to run SLiMBench directly from the command line in the form:

```
python slimbench.py [cmd_list]
```

To run with default settings, no other commands are needed. Otherwise, see the relevant sections of this manual.

IMPORTANT: If filenames contain spaces, they should be enclosed in double quotes: *data="example file"*. That said, it is recommended that files do not contain spaces as function cannot be guaranteed if they do.

2.1.2. Options

Command-line options are suggested in the following sections. General details about Command-line options can be found in the RJE Python Appendices document included with this download. Details of command-line options specific to SLiMBench can be found in the distributed readme.txt and readme.html files. These may be given after the run command, as above, or loaded from one or more *.ini files (see RJE Python Appendices for details).

2.1.3. Running in Windows

If running in Windows, you can just double-click the slimbench.py file and use the menu prompts to navigate through the program. It is recommended to use the **win32=T** option. (Place this command in a file called slimbench.ini.) **Note:** The menu system may not yet be implemented.

2.2. Input

Please refer to chapters 3 and 4 for details of SLiMBench input for dataset generation and benchmarking, respectively.

2.3. Output

Please refer to chapters 3 and 4 for details of SLiMBench output for dataset generation and benchmarking, respectively.

2.3.1. Log Files

The SLiMBench log file records information that may help subsequent interpretation of results or identify problems. Probably it's most useful content is any error messages generated. By default the log file is slimbench.log but this can be changed with the **log=FILE** option. Logs will be appended unless the **newlog** option is used. (See the RJE Python Appendices document for details.)

2.4. Commandline Options

Commandline options are given in the appropriate sections. A full list of commandline options can be found in the readme file, online at bioware.soton.ac.uk or by running:

```
python slimbench.py help
```

3. Dataset Generation

SLiMBench can generate three different types of benchmarking data:

1. **ELMBench.** ELM Benchmarking data based on proteins with known ELM occurrences.
2. **SimBench.** Simulated Benchmarking data using ELM-containing proteins from a given proteome (`randsource=FILE`) as positives (signal) and the whole proteome as background (noise). These are matched by an equal number of random datasets using the same Query protein but taking both "signal" and "noise" from the whole proteome.
3. **RanBench.** Completely random selections from the selected protein input data.

3.1. ELM Datasets

1. Setup
    a. Load ELM class data
    b. Load ELM instance data. Split accession numbers for UniProt xref.
    c. Read in UniProt entries
    d. Generate ELM motif file (*.motifs.txt)
2. Generate basic ELM datasets. For each ELM construct a dataset of the UniProt sequences with an instance of that ELM. Saved in Data/ELM_Datasets/.
3. Perform SLiMMaker Reduction of ELM Datasets (unless `slimmaker=F`). This is performed using SLiMMaker defaults unless otherwise specified. (See SLiMMaker manual and Appendix 5.3). The exception is `iterate=T/F`. SLiMMaker reduction in SLiMBench is iterative, such that the reduced motif produced is found in all retained instances. These are then used to make "reduced ELM" datasets of just the sequences containing the reduced ELM in Data/ELM_Datasets_Reduced/. Reduced ELMs are saved as *.reduced.motifs.txt.
4. SLiMSearch is used to search the reduced ELMs against their datasets to check for the minimum number of UPC (`minupc=X`). The reduced ELM must also meet the minimum IC criterion (`minic=X`).
5. Each retained ELM is used to make a set of datasets with each protein in the dataset selected in turn to be a query (saved in Data/ELM_Datasets_WithQueries/). Each query region masking option is applied at this stage.

3.1.1. Query Region Masking

Masking specific query regions around the motif instance is performed by making the query sequence lower case except for the region specified. This can then be masked by (Q)SLiMFinder/SLiMSearch using *`casemask=Lower`*. The following region masking options are allowed, given as a list of the desired output (each is made):

- **none.** No region masking. (Entire sequence upper case.)
- **site.** Only the motif instance itself is uppercase. This includes internal wildcard positions but no flanking sequence. Multiple instances are all unmasked.
- **flankX.** Each motif instance and X flanking positions each side are unmasked, unless truncated by the start/end of the sequence.
- **winX.** A window of X residues, centred on the motif instance, is unmasked. If the instance is too close to the start/end of the sequences, this window is slid along the sequence to give X unmasked positions (or the full sequence if shorter). In the case of multiple instances, the window is chopped up such that the combined unmasked region should be X. For n instances, a window of X/n is first centred on each instance and extended to include the full instance if necessary. The total number of unmasked positions is then counted, which will frequently be less that X due to instances that are close to each other and/or the sequence start/end. In this case, all windows are grown one position at a time until either X+ positions are unmasked or the full sequence is unmasked.

3.1.2. Datasets for Benchmarking Without Queries

To benchmark SLiM prediction without using queries, the “reduced ELM” datasets in Data/ELM_Datasets_Reduced/ (stage 3) can be used and predictions benchmarked against the reduced ELMs in *.reduced.motifs.txt.

3.2. Simulated Datasets

The Simulation method of SLiMBench (**simulate=T**) takes the reduced ELM patterns as Input and apply the minic filter to them. It will then perform a search against the target proteome (**randsource=FILE**) using rje_slimcore.motifSeq() to apply selected masking options (unless **masking=F**).

Simulated datasets consist of a number of different replicates (**randreps=X**) for each of a number of different signal:noise ratios. The latter are controlled by two parameters: **simcount=LIST** sets the numbers of True Positive proteins for each dataset to contain (in addition to random background), whilst **simratios=LIST** sets the “noise” component in terms of 1:X signal:noise.

Example. *simcount=5,10 simratios=1,2* would create four different dataset compositions with signal:noise counts 5:5 (5/10), 5:10 (5/15), 10:10 (10/20), 10:20 (10/30).

Simulated datasets are constructed as follows:

1. Setup SLiMCore object and the random source (**randsource=FILE**). ELMs are loaded from **elmclass=FILE** in addition to reduced motifs (*.reduced.motifs.txt) made during ELMBench construction and applies minic filter (**minic=X**). Will switch to randomisation if not found. (See 3.3.)
2. Existing files of (masked) ELM-containing proteins will be looked for and re-used if found. These are named genpath/SourceMatch/ELM.SOURCEBASE.fas based on **genpath=PATH**, where SOURCEBASE is the base filename of **randsource=FILE** and ELM is the name of the ELM. The random source will be masked unless **masking=F**.
3. SLiMCore will then be used to find all occurrences of each (missing) ELM in the (masked) proteome and output to files using the motifSeq() method.
4. SLiMSearch is run on each ELM against the (masked) motifSeq files produced by SLiMCore. These results are then loaded into Database object tables.
5. Each ELM is taken in turn and used to generate a number of replicate simulated dataset pairs as set by **randreps=X**. Note that **minic=X** is reapplied at this point, allowing the re-running of the SimBench construction with a more stringent cutoff.
6. Each number of TP occurrences is taken in turn (**simcount=LIST**) and compared to the UPC support of the SLiMSearch results. If there are sufficient UPC (each TP must come from a different UPC) then datasets are generated.
7. For each **randreps=X** replicate, a random query occurrence is selected. This is used to determine both the query protein and also the query region. The same query is then used for all signal:noise ratios. Note that the query occurrence is chosen at random *with replacement*, so the same query could feature more than once. (This means that ELMs with small numbers of occurrences can still generate large numbers of datasets if desired.)
8. For each TP count and each signal:noise ratio, all other sequences are randomly selected independently for each dataset. First, occurrences are selected at random (*without* replacement) and added to the “signal” if they do not share a UPC with an existing signal proteins. This is repeated until the TP set (**simcount=LIST**) is full. The remaining proteins are then selected at random from the remainder of the proteome, i.e. independently of the presence/absence of the ELM or any UPC relationships. This is the “sim” dataset and is named: randir/ELM.sim.rX.pX.nX.REGION.fas, where REGION corresponds to the selected **flankmask=LIST** settings.
9. A matching random dataset is made for each simulated dataset, containing the same query sequence but filled up by random proteins from the whole proteome instead of selecting TP proteins for the “signal”. This is the “ran” dataset and is named: randir/ELM.ran.rX.pX.nX.REGION.fas.

3.3. Randomised Datasets

Randomised datasets (**randomise=T simulate=F**) are generated in the same way as the simulated datasets with the exception that there are no protein sequences selected for containing a specific ELM. A random sequence is chosen to act as the query for any flanking region masking options (**flankmask=LIST**) and a random 10aa sequence within that query is selected to be the motif occurrence of interest. Note that this may not work very well with masked analyses as there is no guarantee that the query region will not be masked out. For randomised datasets, the **randbase=X** is used to set the base for dataset names in place of ELM.sim or ELM.ran.

3.3.1. Uniform frequency distributions

To generate randomised/simulated data without local composition biases, the **seqshuffle=T** function of rje_seqlist (included in libraries/) can be used:

```
python slimbench.py seqin=FILE seqout=NEWFILE seqshuffle=T rename=T
spcode=X newacc=X newgene=X
```

The **NEWFILE** produced can then be used by SLiMBench as the **randsource=FILE** file.

4. Benchmarking Details

SLiMBench benchmarking reads in a set of SLiMFinder and QSLiMFinder results files and benchmarks them against a set of know motifs, usually ELM. This is activated with the `benchmark=T` switch.

The main input is the ELM class file (`elmclass=FILE`), the motif file (`compdb=FILE`) and the list of results files (`resfiles=LIST`, *.csv by default). If no compdb file is used, the default will be to first look for the *.reduced.motifs.txt file generated during benchmarking based on the ELM class file and output path (`genpath=PATH`). If this is not found, the ELM class file itself will be used. If this is not found either, no benchmarking will be performed.

The formatting of the results files is quite important, although they do not *have* to be (Q)SLiMFinder predictions: predictions from other methods will need to be reformatted. The 'RunID' column should contain any information that separates the different SLiM prediction methods, separated by a period. By default, this should be PROGRAM.MASKING. This can be changed using `runid=LIST`. Dataset names should match the format set during SLiMBench dataset generation and toggled between ELM (Motif.Query.Region) and Simulated data (Motif.Query.Region.RType.rRep.pPosNum.nSeqNum) using the `datatype=X` option.

Output file names are determined by `benchbase=X` (default = 'slimbench') and are listed in the corresponding sections, below. Unless `force=T`, existing files at each stage will be read in and used rather than regenerated. (If `force=T`, use `backups=T/F` to control whether existing results are backed up.)

SLiMBench benchmarking involves the following stages:

1. **Setup**. The ELM class and motif comparison databases are loaded.
2. **Load SLiM Predictions.** SLiM prediction results files are loaded and combined. Dataset and RunID data are extracted and combined results saved (*.results.tdt).
3. **CompariMotif analysis of results.** A non-redundant set of all SLiM prediction are used as input for a CompariMotif search against compdb and results output to *.compare.tdt.
4. **SLiM Prediction Ratings.** SLiM Predictions are compared to CompariMotif results and assigned as either:
    a. **True Positive (TP).** The SLiM matches the known motif used to generate the dataset.
    b. **False Positive (FP).** The SLiM does not match a known motif.
    c. **Off-Target (OT).** The SLiM matches a known motif but NOT the one used to generate the dataset (or the dataset is random data).

    Details of SLiM Ratings can be found in Section 4.3. Results are output to *.ratings.tdt.
5. **Production of Summary Tables.** Summary information suitable for conversion to text tables are output to *.summary.tdt.
6. **SLiMBench Assessment.** SLiM prediction ratings are compressed across queries and motifs to provide final benchmark statistics. These are output to *.bymotif.tdt and *.assessment.tdt.

Further details can be found in the following sections.

4.1. Load SLiM Predictions

SLiM Predictions are read from the output files identified by resfile=LIST (wildcards allowed, e.g. resfiles=*.csv). Results are initially read in using the Dataset, RunID and Pattern fields to establish the unique keys for each row. Dataset and RunID are then split up on periods (".") into additional fields. Dataset fields are used for subsequent data compression during Assessment (4.5), whilst RunID fields can be set by the user and will keep different analyses separate.

For ELMBench analysis, Datasets split into Motif, Query and Region. For SimBench analysis, Datasets split into Motif, Type (sim/ran), Replicate, Number of Positives, Dataset Size, Query and Region. (See 3.1 and 3.2 for details of file naming.)

4.2. CompariMotif searches of SLiM Predictions

For the CompariMotif search, all *unique* patterns from the predictions are output into *.patterns.txt. This is then used for a CompariMotif search against the motif file (`compdb=FILE`), which is the *.reduced.motifs.txt file generated during benchmarking based on the ELM class file by default. The CompariMotif normalised IC cut-off is reduced to 0.4 by default, otherwise all CompariMotif defaults are used. CompariMotif results are output to *.compare.tdt and read in for rating SLiM predictions (4.3).

4.3. SLiM Prediction Ratings

Results of the CompariMotif search (4.2) are joined to the full set of SLiM Predictions (4.1) to provide a set of matches for each individual result. These are then converted into ratings based on the motif used to generate the dataset and the motif matched. This is executed in two phases: first identifying the "good" patterns and then dealing with the rest.

4.3.1. Rating True Positive and Off-Target Motifs

The full set of matches is output to *.cm_full.tdt. Matches are then removed that fail to meet the following criteria:

- At least two matched positions (MatchPos ≥ 2).
- The CompariMotif match IC must be at least 1.5 (MatchIC ≥ 1.5), which is approx. equivalent to one fixed and one three-fold degenerate position, or two two-fold degenerate positions.
- The CompariMotif normalised IC must be at least 0.5 (NormIC ≥ 0.5).

Following this filtering, the dataset motif and the matching CompariMotif motif are compared. If they are the same motif, the hit is rated as a True Positive (TP).

One of the recurring observations of SLiM prediction is that, due to the non-independence of motifs and motif-containing proteins, it is sometimes possible to return a different genuine motif than the one used to generate the dataset. These are referred to as "Off-Target" (OT) motifs and could be considered as either True Positives or False Positives depending on the situation. For this reason, SLiMBench rates them separately and outputs extra statistics that treat OT motifs in different ways (see 4.5). To be rated as an OT motif, the match must meet a slightly stricter requirements of MatchIC ≥ 2.5 unless it is an exact match (NormIC=1). This is to stop lots of short, low information content matches from being rated as OT. The exception is that patterns identified as TP elsewhere in the data will still be rated as OT in the final rating (4.3.2).

All the final TP and OT matches are output to *.tp.tdt.

NB. These settings were selected after much staring at CompariMotif results for SLiM predictions and in the interests of keeping a manageable number of commandline options are currently hard-coded into SLiMBench. It would be trivial to convert them into commandline options if desired – please just contact the author. The *.cm_full.tdt file has all the matches, which will allow the user to examine those kept and those removed by the filtering.

4.3.2. Rating True Negatives, False Negatives and False Positives

Following the identification of all the TP and OT predictions, all individual SLiM predictions are rated as:

- **True Positive (TP).** CompariMotif hit to ELM used for dataset construction (4.3.1) in an ELMBench or SimBench "sim" dataset.
- **Off-Target (OT).** The motif pattern was identified as a TP or OT in the CompariMotif rating (4.3.1) but is not a TP *or* the dataset is fully random (RanBench or SimBench "ran" data). Note that a pattern identified as a TP in a *different* dataset will still be rated OT even if it failed the extra OT threshold (4.3.1).
- **True Negative (TN).** No motif pattern was returned for a random dataset (RanBench or SimBench "ran" data).
- **False Negative (FN).** No motif pattern was returned for an ELM dataset (ELMBench or SimBench "sim" data).

- **False Positive (FP).** All remaining SLiM predictions are rated as False Positives.

Counts of the five categories are reported in the log file and results output to *.ratings.tdt. This file has some additional fields created:

- **Enrichment.** Calculation of observed UP support divided by expected UP support.
- **CloudCoverage.** Text summary: “CloudSeq / SeqNum (CloudUP / UPNum)”
- **Coverage.** Text summary: “Support / SeqNum (UP / UPNum)”
- **Support.** Text summary: “No. Occurrences / Support / UP”

4.4. Summary Table Output

A small summary table is output comparing the positive results from each program for each ELM/Region/Masking combination. All FP and OT motifs are dropped from the data, which is then compressed to leave the best TP motif prediction for each Program/Query/ELM/Region/Masking combination. This is then compressed again by Query to leave the best prediction for each Program/ELM/Region/Masking combination and the proportion of Queries returning a motif. Finally, the data is reshaped so that top predictions of different SLiM prediction programs for prediction for each ELM/Region/Masking combination are output side-by-side into *.summary.tdt.

4.5. SLiMBench Assessment

The SLiMBench Assessment method is designed to run a number of different assessments for subsequent comparison to check for possible sources of bias within the data. There are four main options controlling this behaviour (Table 1). For each combination of these parameters, the full assessment pipeline is run once. All the assessments are output to the same file (*.assessment.tdt) with columns ByCloud, ICCut, LenCut and SigCut storing the appropriate settings.

Table 1. SLiMBench Assessment processing parameters.

Option	Description	Default
`bycloud=X`	Whether to compress results into clouds prior to assessment (True/False/Both).	[Both]
`sigcut=LIST`	Significance thresholds to use for assessment	[0.05,0.01,0.001,0.0001]
`iccut=LIST`	Minimum IC for (Q)SLiMFinder results for benchmark assessment	[2.0,2.1,3.0]
`slimlencut=LIST`	List of individual SLiM lengths to return results for (0=All)	[0,3,4,5]

4.5.1. Filtering results on Information Content and SLiM length

For each `iccut=LIST` and `slimlencut=LIST` combination, results are filtered prior to assessment. The latter corresponds to the number of *defined* positions in the predicted SLiM. These could be fixed *or* degenerate. Note that `iccut=LIST` sets the *minimum* IC, whereas `slimlencut=LIST` sets the *exact* length (or all lengths if 0). Any motifs failing to meet the requirements are removed. If all results from a given dataset are removed then it is replaced with a single “empty” result as if the dataset returned no motifs in the original search (e.g. Pattern = “-” and Rank = 0). Otherwise, if the top-ranked motif is removed then the next highest ranked motif is given a Rank of 1.

4.5.2. ELMBench Assessment calculations

The goal of the assessment method is to convert the full set of SLiM predictions into weighted measures of SLiM prediction performance. For this, the data undergoes several cycles of compression and re-weighting so that each ELM contributes equally to the final assessment:

1. In addition to the five SLiM ratings from 4.3, results are given three additional ratings which will be calculated during data weighting:
    - **Sensitivity (SN).** The proportion of TP datasets returning a TP motif.
    - **Dataset False Positive Rate (FPX).** The proportion of datasets returning a FP motif.
    - **Stringent Dataset False Positive Rate (FPXn).** The proportion of datasets returning a FP or OT motif.

Initially, each of these values is set to 0 for each motif prediction with the exception of the motif rating (TP/OT/FP/TN/FN) which is given a score of 1. A result counter, N, is also added with a value of 1.

2. If results are being grouped by cloud then individual motif results are compressed into a single result for each cloud. Each cloud keeps the best single positive rating (TP > OT > FP) and sets the other scores to 0.
3. The SN score is set equal to the TN score and FPX is set equal to FP. FPXn is set as 1 if either FP or OT are 1.
4. Results are compressed by dataset such that each Query contributes equally to the results for a given ELM. N sums the number of results (clouds or motifs) for the dataset. SN, FPX and FPXn take the maximum value, *i.e.* remain 1/0 indicating whether that dataset has returned *any* TP or FP(/OT). The mean of individual ratings (TP/OT/FP/FN) are taken across predictions (if FN, there will be only 1 result!) and so these fields now represent the *proportion of motifs/clouds* with those ratings.
5. Within each dataset, results are compressed across queries to remove any bias due to different ELMs having different numbers of queries. For this, the mean value is taken across all queries. These data are output to *.bymotif.tdt.
6. Finally, results are compressed for each Program/Analysis combination to give mean values for each statistic.
7. **Positive Predictive Value (PPV)** is calculated as TP / (TP + FP). Two additional variants are also calculated: PPVp has OT reclassified as TP, and PPVn has OT reclassified as FP.
8. Results are output to *.assessment.tdt.

4.5.3. SimBench Assessment calculations

SimBench data is processed in a very similar way with a few minor changes. SN and PPV values are calculated using the "sim" datasets only. FPX(n) values are calculated using the "ran" datasets only. Following compression at Step 4, "sim" and "ran" data are combined for each pair of datasets prior to subsequent ELM compression. In Step 6, different signal:noise combinations are kept separate in the final output, enabling the investigation of changing dataset size and/or signal:noise ratios (see 3.2). These are output in the "PosNum" and "SeqNum" columns.

If **simonly=T** then the "sim" datasets are used for all statistics and the "ran" data are essentially ignored.

4.6. Benchmarking without Query sequences

SLiMBench can also be used for slightly simpler benchmarking of ELM datasets without specific query data. For this, it is recommended to run predictions on the "reduced ELM" datasets in Data/ELM_Datasets_Reduced/ and benchmark against the reduced ELMs in *.reduced.motifs.txt. This can be instigated with the **queries=F** option.

Benchmarking without queries is executed in the manner described above with the minor modifications that: (1) Query Regions are not considered (as they do not exist) and dataset names do not contain a Region element; (2) There is no compression across Queries for each ELM. Output statistics and files are otherwise unchanged.

5. Appendices

5.1. Troubleshooting & FAQ

There are currently no specific Troubleshooting issues arising with SLiMBench. Please see general items in the RJE Python Appendices document and contact me if you experience any problems not covered.

5.2. Abbreviations and Glossary

The following terms and abbreviations are used in this manual:

- **ELM.** Eukaryotic Linear Motif. Also refers to the ELM database of SLiMs at www.elm.eu.org.
- **FN.** False Negative. No motif pattern was returned for an ELM dataset.
- **FP.** False Positive. SLiM predictions not matching a known ELM.
- **FPX.** Dataset False Positive Rate. The proportion of datasets returning a FP motif.
- **FPXn.** Stringent Dataset False Positive Rate. The proportion of datasets returning a FP or OT motif.
- **OT.** Off-Target. A real motif returned by an unexpected dataset.
- **PPV.** Positive Predictive Value. The proportion of predicted motifs that are TP. PPV ignores OT motifs. PPVp has OT reclassified as TP, and PPVn has OT reclassified as FP.
- **SLiM.** Short Linear Motif.
- **SN.** Sensitivity. The proportion of TP datasets returning a TP motif.
- **TN.** True Negative. No motif pattern was returned for a random dataset.
- **TP.** True Positive. A real motif returned by the expected dataset.

5.3. SLiMMaker commandline options

SLiMMaker is used during ELM Dataset construction (see 3.1). More details can be found in the SLiMMaker documentation but the main commandline options for ELM reduction are:

- `minseq=X` : Min. no. of sequences for an aa to be in [3]
- `minfreq=X` : Min. combined freq of accepted aa to avoid wildcard [0.75]
- `maxaa=X` : Max. no. different amino acids for one position [5]
- `ignore=X` : Amino acid(s) to ignore. (If nucleotide, would be N-) ['X-']

5.4. Bibliography

Future additions of this manual will update citations and divide this bibliography into references and further reading. More publications can be found at bioware.soton.ac.uk.

- Amanchy R, Periaswamy B, Mathivanan S, Reddy R, Tattikota SG & Pandey A (2007). "A curated compendium of phosphorylation motifs." *Nat Biotechnol.* **25**(3): 285-6.
- Balla S, Thapar V, Verma S, Luong T, Faghri T, Huang CH, Rajasekaran S, del Campo JJ, Shinn JH, Mohler WA, Maciejewski MW, Gryk MR, Piccirillo B, Schiller SR & Schiller MR (2006). "Minimotif miner: A tool for investigating protein function." *Nat Methods.* **3**(3): 175-7.
- Davey NE, Shields DC & Edwards RJ (2006). "Slimdisc: Short, linear motif discovery, correcting for common evolutionary descent." *Nucleic Acids Res.* **34**(12): 3546-54.
- Davey NE, Edwards RJ & Shields DC (2007). "The slimdisc server: Short, linear motif discovery in proteins." *Nucleic Acids Res* **35**(Web Server issue): W455-9.
- Edwards RJ, Davey NE & Shields DC (2007). "Slimfinder: A probabilistic method for identifying over-represented, convergently evolved, short linear motifs in proteins." *PLoS ONE* **2**(10): e967.

- Neduva V, Linding R, Su-Angrand I, Stark A, de Masi F, Gibson TJ, Lewis J, Serrano L & Russell RB (2005). "Systematic discovery of new recognition peptides mediating protein interaction networks." *PLoS Biol.* **3**(12): e405.
- Neduva V & Russell RB (2005). "Linear motifs: Evolutionary interaction switches." *FEBS Lett.* **579**(15): 3342-5 Epub 2005 Apr 18.
- Neduva V & Russell RB (2006). "Dilimot: Discovery of linear motifs in proteins." *Nucleic Acids Res.* **34**(Web Server issue): W350-5.
- Puntervoll P, Linding R, Gemund C, Chabanis-Davidson S, Mattingsdal M, Cameron S, Martin DM, Ausiello G, Brannetti B, Costantini A, Ferre F, Maselli V, Via A, Cesareni G, Diella F, Superti-Furga G, Wyrwicz L, Ramu C, McGuigan C, Gudavalli R, Letunic I, Bork P, Rychlewski L, Kuster B, Helmer-Citterich M, Hunter WN, Aasland R & Gibson TJ (2003). "Elm server: A new resource for investigating short functional sites in modular eukaryotic proteins." *Nucleic Acids Res* **31**(13): 3625-30.