BioV Documentation

A set of programs designed to aid research in the field
of transport protein evolution.

vreddy@ucalgary.ca

http://biov.tcdb.org

Installation

The BioV suite has been tested on Ubuntu 10+ and OS X Leopard and OS X Lion. However, this suite should run on virtually any unix based operating system.

Before we begin, please ensure you have the following dependencies installed.

- Python 2.7
- BLAST (Latest)
- EMBOSS Package
- Matplotlib
- BioPython
- FASTA Package
- Mechanize
- HMMTOP

All of these packages are available from most unix installation managers (except FASTA package & HMMTOP). It is highly recommended that you install them using a repo manager, IE. apt-get or darwin-ports.

Once all of these programs are installed properly and can be accessed from the shell (exist in the $PATH variable), download the BioV suite here.

Extract the folder contents to any folder on your machine. Add this folder path to your $PATH directory and to your $PYTHONPATH directory. To do this you will need to edit your profile file. This path varies in operating systems. For OS X it is found in ~/.bash_profile. And in Ubuntu it is ~/.bashrc.

All BioV scripts have a shebang line pointing to /usr/local/bin/python. If your installation is somewhere else, you should make a dynamic link to this path.

Most errors arise from an incorrectly installed copy of HMMTOP and SSearch36(FASTA Tools). Make sure these executable are in one of your $PATH folders. Also, remember to define $HMMTOP_PSV and $HMMTOP_ARCH in your profile to point to both of HMMTOP's dependencies.

If you did everything correctly, you should be able to execute any BioV program from your shell from any working directory.

TMStats

Filename: tmstats.py
Description: Tabulate and generate a bar graph of TMSs and their occurrences within a FASTA file or TC hierarchy.

Basic Usage: tmstats.py

Usage: `tmstats.py <fasta_file> <title> <output>`

Where 'fasta_file' is the path to your list of FASTAs. 'Title' is the text to display on the graph. 'Output' is the file path to write to.

Example usage:
```
tmstats.py mfs.faa 'MFS TMS distribution' myoutput.png
```

To analyze a TC hierarchy, use the web-based version of TMStats. TCDB is frequently updated, so this is the best solution. TMStats-Web can be found here: http://www.tcdb.org/progs/?tool=tmstats

To use the web based version simply enter any TC number. You many enter multiple TC numbers; just separate by commas. Use the '%' symbol to denote a wildcard in the TCID.

For example, to get stats on all the MFS proteins (2.A.1), you would enter: **2.A.1.%**

Or, to get info on the first TCID of each cluster within the MFS family, you would enter: **2.A.1.%.1**

To combine results of MFS & APC, you would enter: **2.A.1.%,2.A.3.%**

Programmatic Access

You can interface with the TMStats program within python scripts. Here is an example:

```
import tmstats
fasta = 'path/to/fasta.faa'
label = 'My bargraph'
output = './graph.png'
tabs,tms = tmstats.calculate(fasta,label,output)
# output written to ./graph.png
# tabs contains raw statistics
# tms contains TMS locations
```

GSAT – Global Sequence Alignment Tool

Filename: gsat.py

Description: Perform a shuffle based alignment on two sequences.

Basic Usage:
From the terminal type: **gsat.py**

Enter the 'A' sequence and the 'B' sequence when prompted. Make sure each sequence contains no line breaks.
By default, this approach uses several default values that have been determined as the optimal settings for many transport proteins. These settings are: Gap Open Penalty = 8, Gap Extension Penalty = 2, and 500 random shuffles.
This should finish instantly, and return a report with your alignment and the standard score (z-score) at the very bottom.

Alternatively, this program can be run from the official TCDB website, using this link : http://tcdb.org/progs/?tool=gsat

Using GSAT in a programmatic context:
The best way to showcase all of GSAT's features is by example.
Here is a sample code with commentary.

```
import gsat # Import the package
gs = gsat.cmd() # Initialize our GSAT object
gs.asequence = 'ABCD' # First sequence to compare
gs.bsequence = 'VBSC' # Second sequence to compare
gs.gapopen = 8 # NW Gap open cost (default 8)
gs.gapextend = 2 # NW Gap extend cost (default 2)
gs.shuffles = 500 # Number of shuffles to perform (default)
gs() # Begins the actual process

## Now that GSAT has run, lets extract some data ##

# Grab just the needle outfile content.
# This file contains alignment stats for just ONE shuffle.
gs.outfile.seek(0) # outfile contains the handle, rewind it first.
print gs.outfile.read()

# Grab calculated data
print gs.zscore # Prints the z-score, rounded to the nearest whole
number.
print gs.zscorep # Same as above, but not rounded.
Print gs.average # Average NW-bit score from shuffles
print gs.gaps # prints the percentage of gaps in alignment
```

Protocol 1

Filename: protocol1.py

Description: Perform NCBI-PSI Blasts with iterations remotely without downloading a local database. In addition, one may annotate and tabulate statistics about a generated dataset.

Basic Usage:
From the terminal type: **protocol1.py**

You will be prompted for an accession number or a gi number. You may also enter a sequence to BLAST, but make sure it is without the header and contains no line breaks. Next you are prompted to count the TMSs. This will generate an additional bar chart with the number of TMSs and their occurrences. The next prompt is the output path. This can be anything, and protocol1 will generate this folder containing your data. Finally, you will be prompted for a cd-hit cutoff value. This will remove sequences that are too similar to others in the population. This can be a value from 0.4 – 1. This corresponds to 40% - 100% identity. A value of '1' will remove only duplicate sequences. For more information, read the cd-hit user manual.

Below are all the settings that can be applied to Protocol1. The lines in **BOLD** are applied by default, unless explicitly specified otherwise.

```
Options:
  --version   show program's version number and exit
  -h, --help  show this help message and exit
  -q QUERY    Gi/Accession/Sequence to BLAST.
  -i ITERATE  Number of additional iterations to perform. (0)
  -n NUMBER   Number of results to fetch each round (500)
  -e EXPECT   E-Value cutoff (0.005)
  -c CUTOFF   CD-HIT threshold. From 0.4 - 1 (0.8)
  -o OUTDIR   Output folder (p1out)
  --tms       Include this flag to tabulate TMS stats.
  --min=MIN   Minimum sequence lengths to retrieve
  --max=MAX   Maximum sequence lengths to retrieve
```

Advanced usage:
If a user wants to generate a FASTA list of Q7VW14 from a total of 3 iterations. Has a blast expectation value of 0.0003, and wants tabulated TMS stats. In addition, the user wants the retrieved sequences to be between 300-350 aas long. In addition, the user wants 1000 results or less. Here is the command for this circumstance.

```
protocol1.py -q Q7VW14 -i 3 -n 1000 -e 0.0003 --tms --min=300 --max=300
```

When this is done, your files will be in the 'p1out' folder by default.

Protocol1 programmatic access:
Protocol1 can be accessed within other python scripts. However, it also requires the aid of a helper module called 'blast.py'. Blast.py will build a TinySeq XML format file and a raw FASTA file when provided with a list of gi numbers. In this example, we will use protocol1 to generate a list of gis, and then pass this list to our helper function within blast.py

```
import protocol1
import blast

psi = protocol1.psi()
psi.query = 'Q7VW14'
psi.expect = 0.005
psi.results = 500
# Result numbers must be in this list:
# 10,50,100,250,500,1000,5000,10000,20000
psi.init_blast()
psi.iterate()
psi.iterate()
psi.iterate()
# This will perform 3 additional iterations.
psi.fetch_results()
# Done, now generate fasta file with blast tools
blast_tool = blast.tools()
blast_tool.gis = protocol1.gis # Link our gi list
blast_tool.build_xml() # Build TinySeq file.
# Handle can be found in blast.tool.xml_file
blast_tool.build_raw_fasta() # Generate raw FASTA file
# FASTA handle found in blast_tool.raw_fasta
# Both files are temporary handles. They will be lost when the
# program exits. The next 3 lines demonstrate how to save it from
# memory to disk.
import shutil
shutil.copy(blast_tool.xml_file.name,'TinySeq.xml')
shutil.copy(blast_tool.raw_fasta.name,'myfasta.faa')
```

DefineFamily

Filename: define_family.py
Description: Generate FASTA files to represent any TC Family.
Basic Usage: type into the shell: **define_family.py**

Usage: define_family.py FAMILY <P/PSI> OUTPUT

'FAMILY' can be any 3 unit TC ID. For example, '2.A.1'.
Select P or PSI. This corresponds to BLASTP or PSI-BLAST, respectively.
The last option is OUTPUT. This is the FASTA file that will be created.

For this example, we will generate a FASTA file for 2.A.1 using PSI blast, and write it to Family.faa

```
define_family.py 2.A.1 PSI Family.faa
```

Protocol2

Filename: protocol2.py

Description: Find homologs between two FASTA files & generate graphical reports

Basic Usage: In the terminal type: **protocol2.py**

Usage: protocol2.py [options]

```
Welcome to Protocol2. This tool will allow you to rapidly locate homologs
between two fasta files.      'Subject', 'Target', 'Outdir' are the only
mandatory options. --Subject & --Target are used to label items on   your
actual report. EX. protocol2 -s 2.A.1.faa -t 2.A.3.faa -o mydir
--subject='APC' --target='MFS' // Developed by Vamsee Reddy

Options:
  --version         show program's version number and exit
  -h, --help        show this help message and exit
  -s SUBJECT        Path to subject file
  -t TARGET         Path to target file
  -o OUTDIR         Path to output
  --subject=SNAME   Subject Name to appear on report
  --target=TNAME    Target Name to appear on report
  --assign=NUM      TSS Setting :: Number of targets to assign each subject
  --shuffle=RAND    Number of times to shuffle each alignment with GSAT
  --stms=SRESTRICT  Report will contain X TMS subjects. TTMS must be set to
                    work
  --ttms=TRESTRICT  Report will contain X TMS targets. STMS must be set to
                    work
```

The only mandatory settings are -s, -t, & -o. The optional settings are denoted with a double dash (--). The --subject & --target settings are used to annotate sequences within the report. These are useful incase you forget what you were originally trying to compare. The --assign setting is equivalent to the one found in the TSS program. The --shuffle setting is 500 by default and determines how many shuffles GSAT should perform for each alignment. The last two settings are --stms and --ttms. Both of these must be set for either of them to work. For example, if both of these were set to '12', then the report generated will only deal with proteins containing 12 TMSs.

Example Usage:
We have two fasta files (subject.faa, target.faa) that were called 'MFS' & 'Gap Junction' respectively that we are comparing. We are only interested in MFS proteins with 12 TMSs and Gap Junction protiens with 4 TMSs. Here is the command to run for this scenario.

```
protocol2.py –s subject.faa –t target.faa –o comparison --subject='MFS'
--target='Gap Junction'
```

Programmatic Usage

Protocol2 may also be accessed within programs. Doing this will allow a user to generate HTML reports from command line.

```
import protocol2
compare = protocol2.Compare()
compare.subject_file = 'mysubjects.faa'
compare.target_file = 'mytargets.faa'
compare.outdir = 'path/to/outdir
compare.shuffle = 500
compare.assign = 3
compare.subject_name = 'MFS'
compare.target_name = 'Junctional Family'
compare.srestrict = 12
compare.trestrict = 6
compare()
# Results found in specified outdir
```

TSS – Targeted Smith Waterman Search

Filename: tss.py

Description: TSSearch is a heuristic Smith & Waterman search tool that will rapidly find close & distant homologs between two FASTA files.

Basic Usage: Simply type, **tss.py** into the shell to retrieve a list of options.

```
Usage: tss.py [options]

TSSearch is a heuristic Smith & Waterman search tool that will rapidly find
close & distant homologs between two FASTA files.

Options:
  -h, --help      show this help message and exit
  -s SUBJECT      Path to your subject fasta file
  -t TARGET       Path to your target fasta file
  -o OUTPUT       Output file to create
  -m MAX_TARGETS  Targets per subject (Optional, Default: 3)
  -r RANDOM       Number of times to shuffle each sequence (Optional, Default:
                  300)
```

Typically the default settings are fine for most scenarios. The only settings that are mandatory are 'subject', 'target', & 'output'. For example, If a user has two fasta files (subjects.faa, targets.faa) in their current working directory, the syntax for executing a TSS comparison would look like:

```
tss.py –s subjects.faa –t targets.faa –o compared
```

The output file will be a tab-separated text file containing the subject key, target key, and z-score.

TSS Programmatic access

TSS can be executed within other python scripts. A benefit of the TSS wrapper is there is no need to parse the TSV file. A list of tuples can be read directly. Below is an example usage of the TSS script.

```
import tss
compare = tss.compare()
compare.subject = 'mysubjects.faa'
compare.target = 'mytargets.faa'
compare.max = 3
compare.shuffle = 300
compare()
# results are found in compare.results
```

SSearch – Smith Waterman Search

Filename: ssearch.py

Description: SSearch is very much like TSS, but will do shuffle-based alignments for every combination of subjects & targets.

Basic Usage: In the terminal, type: **ssearch.py**

```
Usage: ssearch.py [options]

SSearch will compare every sequence in a subject file to every sequence in a
Another target file by using a Smith Waterman search. Results are returned
after each pair is shuffled #R number of times, and a Z-Score is returned to
determine the quality of the alignment.

Options:
  -h, --help  show this help message and exit
  -s SUBJECT  Path to your subject fasta file
  -t TARGET   Path to your target fasta file
  -r RANDOM   Number of times to shuffle each sequence (Optional. Default:
              300)
  -o OUTPUT   Filepath To write output
```

Example usage:
We have the exact same variables as the TSS programs.

```
ssearch.py –s subjects.faa –t targets.faa –o compared
```

Programmatic Usage

Programmatic access for SSearch is only slightly different than TSS. However, they return the same output format.

```
import ssearch
compare = ssearch.Compare()
compare.subject = 'mysubjects.faa'
compare.target = 'mytargets.faa'
compare.shuffle = 300
compare()
# results are found in compare.results
```

GBlast

Filename: gblast.py
Description: Identify transport proteins from an entire proteome file.

Basic usage: from the terminal type 'gblast3.py'

```
Usage: gblast.py [options]

Welcome to GBlast! Easily identify transporters in entire
Genomes/Proteomes -
By Vamsee Reddy

Options:
  --version   show program's version number and exit
  -h, --help  show this help message and exit
  -i INPUT    Path to genome/proteome file
  -o OUTPUT   Results output name
  --cdd       Include CDD analysis (Takes a while)
```

This program is very straightforward to use. All you need is a FASTA file to analyze. GBlast can resume incomplete an analysis as well.

Including the -cdd flag will fetch any overlapping conserved domains as identified by NCBI's CDD tool. Including this option will take a long time to complete.

Example usage:

```
gblast3.py -i ICH.fsa -o analysis -cdd
```

AncientRep

```
Filename: ancient.py
Description: Find TMS Repeats within a list of homologues.

Usage: Type into the terminal: ancient.py

Usage: ancient [options]

This is the Ancient program. This tool will find very old (and new) TMS repeat
units. Results can be viewed as they become      available by using the
readlive.py command. Alignments can be viewed using the show_alignment.py
command. This tool was created by:     Vamsee Reddy

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -i INPUT              Path to your subject fasta file
  -r REPEAT             Repeat Size
  -o OUTPUT             Output Name

  Optional Settings:
    --min=MIN           Minimum TMS Requirement
    --max=MAX           Maximum TMS Requirement
    --flank=F_LENGTH    Size of hydrophilic padding around TMSs (10 aa)
    --method=METHOD     [1] horizontal, [2] Vertical, [3] Both
    --VSrestrict=SSET   Vertical Subject Restriction.
    --VTrestrict=TSET   Vertical Target Restriction.
    --consecutive       Includes only consecutive targets 'r' TMSs apart
    --fasta_only        Build ONLY Fasta DB(s) (No TMS Searching)
    --threads=THREADS   Number of threads <default:16>
```

AncientRep (AR) is a dense program with lots of features. An analysis can be very computationally intensive; therefore the program will use all processors by default. AR includes two programs for extracting results, and will be described further into this tutorial. Results can be viewed and analyzed before AncientRep is entirely finished running. The first part of this tutorial will describe how to begin searching for repeats. The next part of this tutorial will explain how to interpret results.

Before working through this guide, please read The Bio-V Suite paper (Reddy & Saier, 2012). In specific, read the section on AR. It is important that you understand how AR works, and are familiar with some of the jargon used when discussing this tool.

Step 1) Obtain a FASTA List. This can be done using the protocol1 program or the DefineFamily program.

Step 2) Now you must select a repeat size. By looking at hydropathy plots of representative proteins, perhaps you should have a theory as to what the repeat size is. If your prediction is wrong, the data is still valuable. The results will often reveal the true repeat size. In reality, all repeat size analyses (regardless if they are correct

or incorrect) should all point to the same true repeat. Just make sure this is an integer value greater than '1'.

Step 3) It is highly recommended that you set the --min & --max settings. Ideally these should be the SAME number. Failure to set these can result in many false positive results, and will make it difficult to interpret the results. For example, if you are working with the MFS family, set these both to 12. This setting will only apply to the vertical search method.

Step 4) For almost all searches, the default flank size is sufficient. By default, AR uses 10aa padding. It is best to perform a vertical and horizontal search. By default AR does both. Horizontal results are more convincing when available. Horizontal searches will reveal recent TMS repeats. The vertical approach will find very old (ancient) repeats. Unless you are very confident in what you expect to find, it is best to leave these settings untouched.

Step 5) *Using VSRestrict & VTRestrict.* These two settings allow you to choose exactly which TMSs you want to compare. If for whatever reason you believe that two sets of TMSs are homologous and don't want to conduct an exhaustive search -- this can save time. Such foresight can be derived from interpreting hydropathy plots. For the next few examples, we will be looking at a hypothetical six TMS protein.

Scenario 1: We are certain that TMS 1 & 2 gave rise to 3 & 4. We want to restrict the subject (V**S**Restrict) to "1 & 2" and restrict the target (V**T**Restrict) to "3 & 4". Here is an example command, with this section in bold:

```
ancient.py -i myfastas.faa -o myoutput -r 2 --min=6 --max=6 --
VSRestrict='(1,2)' --VTRestrict='(3,4)'
```

Scenario 2: We want to only compare TMSs 1,2 with 3,4 & 5,6.

```
ancient.py -i myfastas.faa -o myoutput -r 2 --min=6 --max=6 --
VSRestrict='(1,2)' --VTRestrict='(3,4),(5,6)'
```

Scenario 3: We only care about which TMSs gave rise to 4,5. This will compare ALL TMSs **before** TMSs 4,5 to these positions.

```
ancient.py -i myfastas.faa -o myoutput -r 2 --min=6 --max=6 --
VTRestrict='(4,5)'
```

Scenario 4: You have identified TMSs 3,4 as a precursor for something else. This command will compare TMSs 3 & 4 to all the TMSs **after** this pair. Pay attention to the difference with Scenario 3.

```
ancient.py -i myfastas.faa -o myoutput -r 2 --min=6 --max=6 --
VSRestrict='(3,4)'
```

Keep in mind; it is not necessary to set any of these vertical restrictions. Leaving these options blank will instruct the program to perform all *logical* TMS combinations.

Step 6) *When to use the --consecutive* flag. The way AR works is as follows: the subject frame selects 'r' TMSs from the beginning of a protein. For this example, we will use an r value of '2'. The subject frame selects TMSs 1,2. The target frame will select 3,4 and will continue to slide in increments of '1' until it reaches the end of the protein. Ultimately, TMSs 1,2 will be compared to (3,4),(4,5),(5,6). If the --consecutive flag is enabled, the target frame will slide in increments of 'r' (2). With this flag enabled, TMSs 1,2 will be compared to (3,4),(5,6). Using the --consecutive flag will increase the speed of AR. However, this should only be used if you are confident in your 'r' size and understand that the family of transporters you are working with is well conserved. Many proteins may have experienced some type of insertion or deletion after the repeat event. These events will make the --consecutive flag unreliable. Here is an example command for the --consecutive flag.

```
ancient.py -i myfastas.faa -o myoutput -r 2 --min=6 --max=6 --
consecutive
```

Step 7) *When to use the --fasta_only flag.* If this flag is included at the end of any command, AR will just built a FASTA database of all the individual TMSs without actually performing any comparisons. This is useful if you need to build a BLAST DB that consists of only TMSs. Example:

```
ancient.py -i myfastas.faa -o myoutput -r 2 --min=6 --max=6 --
fasta_only
```

Step 8) *When to use --threads.* This option is for advanced users. By default, AR uses the maximum number of threads available to the machine. This corresponds to the number of processors the machine has. There is no benefit to lowering the default value, unless you need to reserve x number of processors for another task. Example:

```
ancient.py -i myfastas.faa -o myoutput -r 2 --min=6 --max=6 --threads=5
```

This is all the information you will need to get AR running. The rest of this tutorial will explain how to interpret your AR results.

Interpreting Results

Understanding AR is straight forward, once you are familiar with all the ropes. There will be a lot of results, so the trick is to find results with good SD scores (10 SDs) with low percent gaps (typically less than 15%). These standards are not set in stone. An acceptable percent gaps can fluctuate greatly. It depends on where the gaps are. If the gaps are in the hydrophobic region, it is likely that the TMSs being compared are not aligning. If it is in the hydrophilic portion, then there is a chance that the actual TMSs are aligned, and you have found a potential repeat.

The first results to examine are the horizontal results. From your terminal, type 'cd OUTPUTFOLDER' and replace OUTPUTFOLDER with the directory you

originally wrote. Type 'ls' to see the contents of the folder. There should be a file called 'horizontal.txt' and several folders with numbers in them. Use the readlive.py tool to display your results. The format for executing the readlive.py script is as follows:

```
readlive.py <filename> <all/consec> <min_sd>
```

'Filename' corresponds to the file we are analyzing. In this case, it is horizontal.txt. The next parameter is 'all' or 'consec'. Choosing 'consec' will make sure that the target TMSs are consecutive integers to the subject TMSs compared. For example, the script will retrieve comparisons between TMSs 1,2 vs 3,4 and TMSs 3,4 vs 5,6. It would never retrieve TMSs 1,2 vs 4,5. Consecutive results almost always make evolutionary sense. However, no data should be ignored. The final parameter is 'min_sd' This is the minimum SD scores the program will retrieve. This is the full command to run:

```
readlive.py horizontal.txt all 10
```

If no results are returned, try lowering the min_sd value by increments of 1 until results are obtained. When results are available, they will be sorted by SD and % gaps, with the best scores displayed at the bottom. Scroll up until you find a potential repeat. To view the actual alignment, copy the entire line to the clipboard. Then type into the shell: '`show_alignment.py`'. When prompted, paste the line into the terminal, and hit enter. The full alignment should be displayed. Verify that none of the TMSs are gapped out. If you are happy with the alignment, then optimize it. This will increase/decrease the flank size until the alignment returns the highest SD. When prompted for the original FASTA file, enter the file path (or just drag & drop) the original FASTA file that was given to AR. When this optimization process is complete, two sequences will be returned. These are your repeat units.

If despite lowering your min_sd value, no good results were found, then you move onto the vertical search results. Assuming we used an 'r' value of 2, with a 6 TMS protein, the following folder should exist: 1-2, 2-3, 3-4, 4-5, & 5-6. Start at the lowest TMSs. Type into your shell: 'cd 1-2'. Once you have entered that folder, repeat the exact same process as you did for analyzing horizontal repeats. The only difference is that the file to analyze is called vertical.txt not horizontal.txt.

Programmatic Interface with TCDB

This section will instruct you on how to use the TCDB module to interact with the TCDB.org website by example

First we start by importing the module named tcdb.py

```
import tcdb

# How to retrieve a FASTA from TCDB using an ACC
fasta = tcdb.acc2fasta('MYACC') # Returns a FASTA

# Get a list of accessions or SeqIO Object
# of FASTAS to define a family
accs = tcdb.define_family('2.A.1') # list of accs
fastas = tcdb.define_family('2.A.1',True) # SeqIO object

# Use a local copy of the TCDB BLAST DB. Including this
# Line will automatically update old DB's over 5 days, and
# Will build the DB with BLAST. File will always be found
# In $HOME/db/tcdb
tcdb.use_local() # That's it.

# Get a Family abbreviation from a TC Family digit.
Names = tcdb.Names()
abvr = Names.get_family_abr('2.A.1') # Contains abrv.

# Get potential substrate from TCID. This feature is
# Still in beta mode, so there may be errors in
# Substrate determination.
Substrates = tcdb.Substrates()
molecules = Substrates.get_tcid_substrates('2.A.1.1.1')
# molecules contains a list of potential substrates
```