Jonathan Borowsky

7/1/22

Cryptic pocket filtering instructions

Note that these instructions are for an lsf queue and assume that the nodes on which the scripts are run have internet access such that the scripts can run commands such as *wget* to obtain data from the internet.

1. Set up an anaconda python environment containing:
   1. os
   2. xml.etree.ElementTree
   3. mdtraj
   4. numpy
   5. csv
   6. scipy
   7. itertools
   8. re
   9. operator. itemgetter
   10. sys
   11. scipy.spatial.distance
   12. Bio.Blast
   13. Bio.Blast.NCBIXML
   14. from time import perf\_counter
2. Install NCBI BLAST+ and download the pdbaa sequence database
3. Install PyMOL (this is only needed for manual filtering)
4. Make a copy of /project/bowmanlab/borowsky.jonathan/FAST-cs/protein-sets/cryptic-pocket-filtering-3/
5. Remove the contents of:
   1. log-files/
   2. iofiles-blast/ except for the every-part-a/ and every-part-b/ folders and their contents

every\_part\_a contains the unzipped part A of MOAD from <https://bindingmoad.org/files/biou/every_part_a.zip> and every\_part\_b contains the unzipped part B from the analogous link.

* 1. iofiles-pdb/ except for the generated-submission-scripts/ folder
  2. filtering-output/
  3. both subfolders of iofiles-manual/

1. In the scripts directory:
   1. update the *directory* variable in all scripts which have it to point to the new location of /project/bowmanlab/borowsky.jonathan/FAST-cs/protein-sets/cryptic-pocket-filtering-3/
   2. Update paths to individual scripts in .sh and .bsub scripts
   3. Update paths to your anaconda installation in your .bsub scripts
   4. Make sure the *debug* variable in filter-blast2pairs.py is set to 0

To run the filtering pipeline run the following in order (from the scripts directory). Jobs spawned by each command require the output from the previous command as input.

1. **bsub < blastp-submit.bsub**

This submission script runs blastp-moad2pdb.py

1. **auto-filter-noarray.sh**

This bash script makes copies of the filter-submit-noarray.bsub submission script with the index\_arg# text replaced with actual numbers and then runs all of the copies. Each such copy runs filter-blast2pairs.py with indices directing it to process a specific subset of the blast hits. I tried using an array job but it kept putting all the jobs on one node which caused them to run extremely slowly.

* 1. If any of the filter-blast2pairs.py jobs crashes partway through, filter-blast2pairs.py can be debugged by setting the *debug* variable to 1 and the *rstart* variable to the index of the problematic structure. Possible issues include differences between binding MOAD, MDTraj, and/or RCSB PDB ligand names, as well as attempts to process multiconformer x-ray crystal structures. A new job to complete the interrupted filtering task can then be started using filter-submit-manual.bsub with the starting and ending indices adjusted to match the remaining proteins (and with *debug* set back to 0).

1. **python compile-automatic-filtering-results.py**

This script collects the individual apo-holo pair output files generated by filtering the BLAST hits, assembles them into a single numpy object, sorts them in order of descending RMSD, and saves the result to filtering-output/. It can be run before all of the jobs launched above have completed and will produce an equally incomplete results list. Update the *serial* variable if you generate multiple saved output files in this manner to keep track of them.

1. **python forward-reverse-pocket-separate.py**

This script divides the output of compile-filtering-results.py by pocket direction and re-ranks them in order of descending cryptic pocket RMSD.

Manual filtering

1. Mount the filesystem containing the working directory on your machine so that PyMOL can read it.
2. Open PyMOL. In the PyMOL command line, enter **run path/to/cf-new-checkstruct-savepockets-groupresis-2022.py** and follow the instructions it prints.

Assembling manual filtering output into a csv file and an npy file:

1. **python compile-automatic-filtering-results.py**

Checking sequence identity between the resulting proteins:

1. **python sequence-identity-check.py**