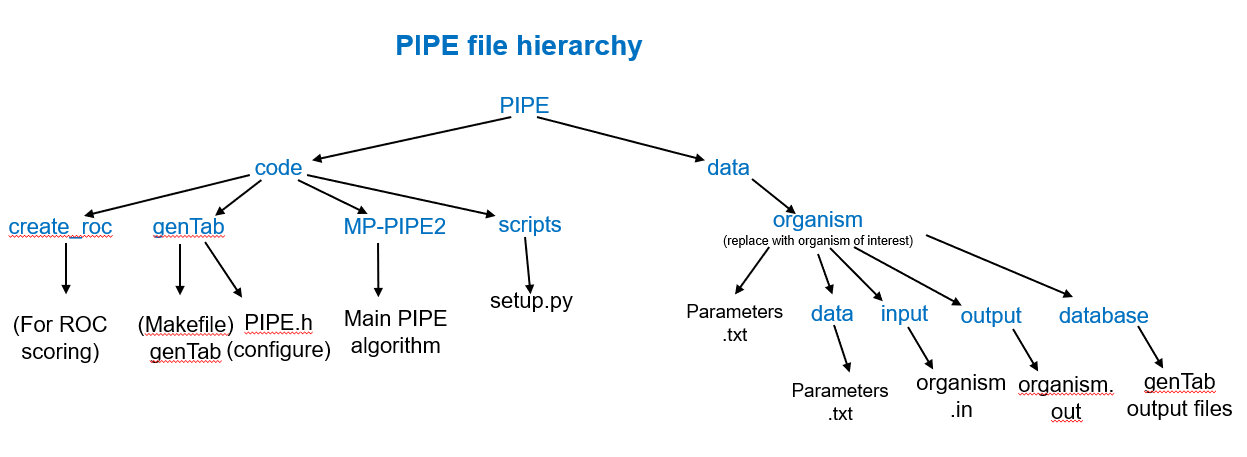
**PIPE documentation** 

**If you are on sharcnet / graham put the PIPE folder in your scratch directory to begin!**

**For example: /scratch/calvinj/PIPE/' is the working pipe version**

**You can always check what directory you are in by typing: “pwd” (meaning print working directory) the exact output you receive from this can be used within modifications to the code and within command line arguments**

**Step 1: Add proteins of interest or interactions of interest:**

**To this file for novel protein sequences:**

**PIPE/PIPE4/data/protein\_sequences.txt**

**Sample protein\_sequence.txt (tab delimited)**

P21192 MDNVVDPWYINPSGFAKDTQDEEYVQHHDNVNPTIPPPDNY…

P46993 MTTLASSIEHKTKHLAAPFENDENPWMKKYCCQCKSCKMS…

P47117 MSYLNNPAVVMDNGTGLTKLGFAGNDSPSWVFPTAIATAAP…

P22768 MSKGKVCLAYSGGLDTSVILAWLLDQGYEVVAFMANVGQE…

P29311 MSTSREDSVYLAKLAEQAERYEEMVENMKTVASSGQELSVE…

Q06333 MQDNSSHSRESASAGDDPLGIDKLTVDYDYLLYKMRDYVQS…

**To this file for novel protein interactions:**

**PIPE/PIPE4/data/protein\_pairs.txt**

**Sample protein\_pairs.txt (tab delimited)**

P47112 P32833

P32585[TAB]P19659

P40013 P09733

P36122 Q12114

P36053 P19454

P15303 Q01477

**Pitfalls and concerns:**

**Large basepair proteins (over 30k basepairs) can lead to overflows**

**Integer overflows can also result from studying very large proteasomes (such as soy)**

**Step 2: Configure GenTab – Run setup-yeast\_gentab.py:**

**Set parameters for PIPE in genTab\_org.txt**

**PIPE4/data/genTab\_org.txt**

**default is:**

**1 // number of species**

**1000 // number of proteins**

**1 // number of species to generate a landscape for**

**0 // index of species 1**

**1: <Number of species>**

**2: <Number of proteins for species 1>**

**: ...**

**: <Number of proteins for species n>**

**: <Number of species we want tot generate files for>**

**: <Index of species 1>**

**: ...**

**: <Index of species n>**

**Set parameters for PIPE in PIPE\_org.txt**

**PIPE4/data/PIPE\_org.txt**

**1 <Number of species>**

**1 <Number of training datasets>**

**0,0 <index of training dataset of species>, <index of training dataset of species>**

**Note:**

Ensure full read/write/execute permissions in the PIPE directory, example:

chmod -R ug+rwx PIPE (recursive, user, group, read, write & exec for the working dir)

if this doesn’t work may need to do chmod -R ugo+rwx working (user, group, other)

**Open:**

path\_to\_PIPE/PIPE/PIPE4/setup-yeast\_gentab.py

**Set the absolute path of “base\_dir”, example:**

base\_dir = “path\_to\_PIPE/PIPE/PIPE4”

**NOTE:** Do not include a trailing slash after PIPE4

**In the PIPE/PIPE4 Directory Run setup-yeast\_gentab.py by typing:**

python setup-yeast\_gentab.py

**NOTE:** This creates a file “protein\_pairs\_index.txt” This file is required later when running genTab. The file is in path\_to\_pipe/PIPE/PIPE4/data

**Step 3: Run GenTab:**

In this step the database folder will be populated with .db files later utilized by PIPE.

**Clean up previous compiled versions and compile a new version**

Go to path\_to\_PIPE/PIPE/PIPE4/code/genTab

**Ensure the output directory of GenTab exists, known as: “database”**

**Example dir is path\_to\_PIPE/PIPE/PIPE4/database**

by typing in:

“mkdir database “

(make sure to create this directory within path\_to\_pipe/PIPE/PIPE4)

**RAM requirement:**

Number of proteins \* length of longest protein \* 8 bytes \* 2 (for redundant array)

To convert to gigabytes divide by 1024^3

Theoretical Example: 40,000 proteins \* 20,000 amino acids \* 8 bytes \* 2 = 12 gb of ram

Experimental Example: 20,500 proteins \* 8,900 amino acids was 1.65 GB of RAM (theoretical is 2.7 GB)

**CPU Runtime:**

Gentab should scale linearly across number of cores, and across execution time. Setting up gentab for 20,500 proteins across 8 cores took about 16 hours, so scale up and down according to cpu cores and number of proteins

11,000 proteins across 20 cores took 2 hours

**Storage requirements:**

20 GB of /database files were generated with those 20,500 proteins. This also scales linearly with number of proteins assuming the same average protein length (this was for humans)

**The arguments required to run genTab are provided here**

0 is the directory location of the compiled genTab

1 is the protein sequences file protein\_sequences.txt (PROTEIN\_SEQ\_FILE)

2 is the protein database directory (to be populated by genTab) (DBDIR)

3 is the genTab\_org.txt settings file (ORG\_SETTINGS)

4 protein\_pairs\_index file created by the convertPairs.pl script (PAIRS\_FILE)

**Example Carleton research compute services) (8 cores, unspecified runtime and ram per CPU**

mpirun -n 8 /home/"YourNameHere"/PIPE/PIPE4/code/genTab /home/"YourNameHere"/PIPE/PIPE4/data/protein\_sequences.txt /home/"YourNameHere"/PIPE/PIPE4/database /home/"YourNameHere"/PIPE/PIPE4/data/genTab\_org.txt /home/"YourNameHere"/PIPE/PIPE4/data/protein\_pairs\_index.txt &> output.out & disown

PIPE version 3 GenTab:

mpirun -n 6 /home/calvinjary/PIPE/code/genTab/genTab /home/calvinjary/PIPE/data/organism/data/protein\_sequences.txt /home/calvinjary/PIPE/data/organism/database &> output.out & disown

**Example (Sharcnet compute Canada) with relative path lengths**

srun -t 30:00 --ntasks 24 --mem-per-cpu=512M --account=”accountName” \

./code/genTab/genTab \ ./data/protein\_sequences.txt \ ./database \ ./data/genTab\_org.txt & disown

Note: if you get the error: “ERROR: Lines in /home/calvinjary/PIPE/data/organism/data/protein\_sequences.txt longer than buffer (24277 byte)”  
you have underestimated the max length of your biggest protein in the genTab/PIPE.h file

If you get a segfault (segmentation fault) you need to add full linux permissions to the PIPE folder with:

sudo chmod -R ugo+rwx PIPE

**Step 4: Configure PIPE:**

**Open:**

PIPE4/setup-yeast\_mp-pipe.py

**Set the absolute path of “base\_dir”, example:**

base\_dir = path\_to\_PIPE/PIPE/PIPE4/

**NOTE:** Do not include a trailing slash after PIPE4

**In the PIPE/PIPE4 Directory Run setup-yeast\_gentab.py by typing:**

python setup-yeast\_gentab.py

**NOTE:** This script sets necessary PIPE parameters including: MAX\_DB\_FILE, NUM\_SET\_BITS, MAX\_PROTEIN\_LEN, MAX\_NEIGHBOURS. It also recompiles MP-PIPE2 .

**Creating Organism.in File**

**Open:**

PIPE4/create\_input\_files.py

**Set the absolute path of “base\_dir” example:**

/home/”YourNameHere”/PIPE/PIPE4

**NOTE:** Do not include a trailing slash after PIPE4

**In create\_input\_files.py specify the range of protein sequences you would like to create input files for, example for all-to-all intraspecies for a yeast protein\_sequence.txt file 6049 long:**

jobs = [('yeast-yeast', 1, 6049, 1, 6049)]

Inter-species example where protein\_sequences.txt contains 23594 human sequences followed by 9 HIV sequences:

jobs = [('hs-hiv', 1, 23594, 23595, 23603)]

**create\_input\_files.py is a Python 3 script. To load Python 3, run:**

module load python/3.6

**Then run:**

python create\_input\_files.py

Note: be sure to execute any script to generate this comparison table in a linux environment (same environment to run PIPE) as a Windows environment will differently encode the file and cause errors

**RAM requirement:**

This is going to be half of the /home/calvinjary/PIPE/data/organism/database directory

(the gentab directory) for any run that uses all proteins. So both all vs all and 1 vs all will use 10 GB of RAM if the /database directory is 20 GB

**CPU Runtime:**

Gentab should scale linearly across number of cores, and across execution time. There is one master thread and one slave thread. So for a 20 core CPU you would set n = 10 and this would use up all 20 cores (meaning 20 threads) Example runtime for 50 proteins vs 20,500 proteins was 3 hours using 3 threads (1 master thread and two threads using 10 GB of RAM each)

**Storage requirements:**

The organism.out file was 150 MB for a 50 vs 20,500 protein run, (about 1 million comparisons) and this file should scale linearly

**Step 5: Run PIPE:**

**Open:**

/home/”YourNameHere”/PIPE/PIPE4/generate\_cedar\_submissions\_yeast.py

**Set the absolute path of “data\_dir”, example:**

data\_dir = ‘path\_to\_pipe/PIPE/PIPE4/’

**In the PIPE/PIPE4 Directory Run generate\_cedar\_submissions\_yeast.py by typing:**

python generate\_cedar\_submissions\_yeast.py

**Open:**

PIPE/PIPE4/autosubmitter\_cedar.py

**Set the absolute path of to the PIPE4 directory:**

base\_dir = 'PATH\_TO\_PIPE/PIPE4/'

**NOTE:** Include a trailing slash after PIPE4

**In the PIPE/PIPE4 Directory Run autosubmitter\_cedar.py by typing:**

Python autosubmitter\_cedar.py

**NOTE:** This script submits PIPE jobs and creates landscapes

**Output of PIPE**

/home/"YourNameHere"/PIPE/PIPE4/landscapes/yeast-yeast

The column headings are explained in the header comment of the pipe c file.

But it should be pipe score, matrix max, run time, Sim score 1, Sim score 2

**Step 6: Create an ROC Curve:**

Go to the PIPE/code/create\_roc/ directory

Run the python script that will create random (assumed negative) interactions. The program is called:

create\_random\_ID\_pairs.py   
and it accepts these inputs:

known\_ID\_pairs\_file /home/calvinjary/PIPE/data/organism/data/protein\_pairs.txt

total\_num\_proteins 11138

num\_pairs\_to\_produce 10000

output\_file calrandompairs.txt

type in this example command:

python2 create\_random\_ID\_pairs.py /home/calvinjary/PIPE/data/organism/data/protein\_pairs.txt 11138 1000000 calrandompairs.txt & disown

this completes the create random ID pairs section.

now type in “make clean” and then “make” following that

/home/calvinjary/PIPE/data/organism/data/protein\_pairs.txt

"Usage: %s <pos file> <#pos> <neg file> <#neg>\n", argv[0]);

mpirun -n 10 /home/calvinjary/PIPE/code/create\_roc/roc /home/calvinjary/PIPE/code/create\_roc/proteincal.csv 48106 /home/calvinjary/PIPE/code/create\_roc/calrandompairs.txt 1000000 > output.out & disown

**Step 7: LOOCV:**

Confirm you have gnuplot with the command:

gnuplot --version

dependencies required for gnuplot:

sudo apt-get install python2

sudo apt-get install python3

sudo apt-get install python-numpy

sudo apt-get install python-scipy

sudo apt-get install python-matplotlib

sudo apt-get install gnuplot

sudo apt-get install cclib

sudo apt-get install libnss-mdns

sudo apt-get install gnuplot

confirm gnuplot is installed with the commands:

gnuplot –version and also: which gnuplot

sudo apt-get install cclib --upgrade

in the /home/calvinjary/PIPE/code/scripts folder you will run the file loocv.py but first open it and:

A- At line 19, Change the local\_dir to the directory where the PIPE folder is. for me it was “/home/calvinjary/PIPE” so I put “/home/calvinjary/"   
        B- At line 20, change the remote\_dit to the same directory so “/home/calvinjary/" again  
        C- At line 28, change organisim\_name to the name of the organism PIPE ran.

On line 124 and 137 replace the

str(num\_mp\_pipe\_hosts)

With how many threads you want PIPE to run on. Keeping in mind pipe.c runs on 2 threads. So if you have 20 cores available set str(num\_mp\_pipe\_hosts) to str(9) and this will run on 18 cores with 1 additional master core, so you will use 19.

In the same /home/calvinjary/PIPE/code/scripts folder open the update.py script and comment out the bottom two sections (for loops), which are for distributed clusters. they include both the “for machines in machines” loops

Open the

/home/calvinjary/PIPE/code/MP-PIPE2/PIPE\_hosts file

Make sure all of the nodes including server and desk and node are commented out with a #. Make sure localhost is the only node not commented out. Example:

localhost

#server

#desk01

#node01

Now you can run the LOOCV script with the command:

python2 loocv.py > output.out & disown

and the results will be in the /organism/data/LOOCV folder

**Step 8: Deep-PIPE Sites:**

On a compute Canada Cluster, create a python virtualenv:

module load python/3.8.2  
virtualenv --no-download ~/SYSC4907-pt  
source ~/SYSC4907-pt/bin/activate  
pip install --no-index --upgrade pip  
pip install --no-index -r requirements.txt

GENERATE MASKS

In Deep-Pipe-Sites/preprocessing/data:

python mask\_pipeline\_Filtered.py

This generates a pkl file with the masks in Deep-Pipe-Sites/preprocessing/data

CREATE PSSMS

Move data\_3\_27 and models folders into Deep-Pipe-Sites, as Deep-Pipe-Sites/data and Deep-Pipe-Sites/models

Then load blast:

module load StdEnv/2020  
module load gcc/9.3.0  
module load blast+/2.11.0

in preprocessing, load fastas:

mkdir pssm  
cd pssm  
cp '../../data/uniprot-proteome UP000002311.fasta' .

load db:

mkdir swissprot  
cd swissprot  
/cvmfs/soft.computecanada.ca/easybuild/software/2020/avx2/Compiler/gcc9/blast+/2.11.0/bin/update\_blastdb.pl --decompress swissprot

Split fasta into fastas:

cd ..

awk -F "|" '/^>/ {close(F) ; F = "fastas/"$2".fasta"} {print >> F}' 'uniprot-proteome UP000002311.fasta'

Make output directory for PSSMs

mkdir pssms

Then create command to run PsiBlast on each and every AA sequence:

cd fastas  
for file in \*.fasta; do echo "./runPsiBlast.sh '$file'"; done >> blastall.cmds  
sbatch blastall.sh  
./blastall.sh

PREPROCESSING

For this, we are using preprocessing.py, but preprocessing\_log.py can be used as well with preprocessing\_log\_batch.sh, which contains an additional parameter to take the log of the PIPE landscape channel

Set correct paths in preprocessing.py and preprocessing\_batch.sh, then batch the job:

cd ../..  
sbatch preprocessing\_batch.sh

Confirm job completed by inspecting .out file for the job

EVALUATE ON TEST SET

cd ../predict

Start an interactive job

salloc --account=def-jrgreen --gres=gpu:1 --cpus-per-task=16 --mem=32000M --time=1:00:00

source ~/SYSC4907-pt/bin/activate  
python predict.py