Oligo Miner

This repository contains the code for the OligoMiner tool.

If you are looking to use probe sequences that we have already generated for various genome assemblies (hg19, hg38, mm9, mm10, dm3, dm6, ce6, ce11, danRer10, tair10), you can download those on our website. If you would like to run the OligoMiner tool yourself, please see below for instructions.

Please remember to cite our pre-print:

OligoMiner: A rapid, flexible environment for the design of genome-scale oligonucleotide in situ hybridization probes Brian J. Beliveau, Jocelyn Y. Kishi, Guy Nir, Hiroshi M. Sasaki, Sinem K. Saka, Son C. Nguyen, Chao-ting Wu, Peng Yin bioRxiv 171504; doi: https://doi.org/10.1101/171504

Note about operating systems

OligoMiner is a set of command-line scripts developed on Python 2.7 that can easily be executed from a Bash Shell.

If you are using standard Linux or Mac OS X sytsems, we expect these instructions to work for you. If you are using Windows, we recommend downloading Cygwin and running the instructions through that environment, but unfortunately we cannot guarantee that these instructions will work for you.

Also note that if you're on a Mac, you will need a C compiler installed (for Biopython, NUPACK, Jellyfish, and potentially the alignment program you choose to use, e.g. Bowtie2). You should download Xcode for this purpose.

Installing OligoMiner dependencies

First, you will need to download all dependencies, which include Python (developed on Python 2.7), NumPy (version 1.8.2+), SciPy, and BioPython. We recommend doing this inside of a Anaconda or Miniconda environment (see step 1 below).

You will also need a stand-alone sequence alignment tool such as Bowtie2.

If you would like to use the optional script to evaluate of your probes to adopt secondary structures, you will need NUPACK.

If you want to use our machine learning algorithm to screen probes, you will also need the scikit-learn (version 0.17+) package. If you want to screen probes for the presence of high-abundance k-mers, you will need Jellyfish.

Installing Python and other required dependencies

We recommend using Anaconda to install new Python modules, as it will automatically install all required Python dependencies in one go (see step 1). If you prefer to do it all without a virtual environment, we also provide instructions below.

1. [Optional but recommended] Download the Python 2.7 version of Anaconda. This will allow you to quickly and easily set up your Python environment using the `conda` commands below.
   ```
   conda create --name probeMining numpy scipy biopython scikit-learn
   ```
   This creates a virtual Python environment called "probeMining" (you can change the name if you want). Now, anytime you want to run Python scripts from your terminal, you can just run:
   ```
   source activate probeMining
   ```
   ... which activates your environment and allows you to run this Python environment that already has required library dependencies installed. to deactivate this environemnt, simply run:
   ```
   source deactivate
   ```
2. *If you're using Anaconda (step 1), then Python is already installed and you can skip this step.* Install Python. We developed this on Python 2.7 and recommend you use OligoMiner with this version. You can find some instructions on installing Python in this PDF document on our website.
3. *If you're using Anaconda (step 1), then NumPy, SciPy and Biopython are already installed and you can skip this step.* Install Python libraries: NumPy, SciPy and Biopython:
   ```
   pip install numpy # versions 1.8.2+
   pip install scipy
   pip install biopython
   pip install scikit-learn # versions 0.17+
   ```
   If you're having trouble executing these commands on a server, one problem may be that you don't have root access. If this is the case, try adding the `--user` argument to the end of the `pip` command to install as a local user.
4. You'll need a genome alignment tool to screen your oligos against your genome of interest. We recommend Bowtie2. If you are using Anaconda, you can easily use bioconda to install Bowtie2. First set up the bioconda channels:
   ```
   conda config --add channels r
   conda config --add channels defaults
   conda config --add channels conda-forge
   conda config --add channels bioconda
   ```
   Next, install Bowtie2:
   ```
   conda install bowtie2
   ```
   After installing Bowtie2, you'll need to build a genome index. We recommend you use an unmasked sequence to build the index. E.g. for hg38:
   ```
   bowtie2-build hg38.fa hg38
   ```

Recommended installation:

1. *If you're using Anaconda (step 1), then scikit-learn is already installed and you can skip this step.* Install scikit-learn:
   ```
   pip install -U scikit-learn
   ```
   If you're having trouble executing these commands on a server, one problem may be that you don't have root access. If this is the case, try adding the `--user` argument to the end of the `pip` command to install as a local user. (If you're using an Anaconda environment forgot to include scikit-learn in the original Anaconda environment creation, you can add it with `conda install scikit-learn`.).
2. If you want to use the `structureCheck.py` script, you need to download and compile the NUPACK source code. You will need to register your email address to receive login credentials before downloading the source code. Once downloaded, you will need to navigate to the

directory where the code is located (there should be a `Makefile` in this directory). For example, if the source code is in the directory `/Path/To/NUPACK/nupack3.0.4`, then do: `cd /Path/To/NUPACK/nupack3.0.4`
 Once you're inside the NUPACK directory, compile the executables with your C compiler by running: `make`
 This should create executable files for many NUPACK functionalites that can be used for secondary structure evaluation, found in `/Path/To/NUPACK/nupack3.0.4/bin'. (See the provided NUPACK user manual for additional information on these executables). You will need these executables within your path for our `structureCheck.py` script to work. You can do this with:
```
export PATH=$PATH:/Path/To/NUPACK/nupack3.0.4/bin
```
 You will also need to set the 'NUPACKHOME' environmental variable: `export NUPACKHOME = /Path/To/NUPACK/nupack3.0.4/`
 NOTE: If you don't want to re-execute these `export` commands every time you open a new terminal, you will need to add the following lines to your `~/.bash_profile` or `~/.bashrc` files:
```
PATH=$PATH:/Path/To/NUPACK/nupack3.0.4/bin
export PATH

NUPACKHOME = /Path/To/NUPACK/nupack3.0.4/
export NUPACKHOME
```
3. If you want to use the `kmerFilter.py` script, you will need Jellyfish. As with Bowtie2, this can easily be installed using bioconda: conda install jellyfish If instead you want to build Jellyfish from source, you will need to navigate to the directory where the code is located, where there should be a `Makefile`. For example, if you the source code is in directory /Path/To/Jellyfish/jellyfish-2.2.6, then do: `cd /Path/To/Jellyfish/jellyfish-2.2.6`
 Once here, compile with: `make`
 NOTE: If you are working on a server environment without root access, you may need to instead type:
```
/.configure --prefix=$HOME
make
make install
```
 And finally, add it to your path: `export PATH=$PATH:/Path/To/Jellyfish/jellyfish-2.2.6`
 As in the previous step, if you don't want to have to run this command every time you open the Terminal, then you should add the following to one of your `~/.bash_profile` or `~/.bashrc` files:
```
PATH=$PATH:/Path/To/Jellyfish/jellyfish-2.2.6
export PATH
```
 Now, you'll also want to build a JF file for your genome of interest, which we also recommend using unmasked sequences for. We also recommend writing the output file as 1-bit (max k-mer count 255, any occuring more reported as '255') and not reporting k-mers that only occur once. E.g. for 18-mers in hg38: `jellyfish count -s 3300M -m 18 -o hg38_18.jf --out-counter-len 1 -L 2 hg38.fa`

Running OligoMiner locally

To make sure all of your dependencies are set up properly, below we will run you through the pipeline using some small example datasets.

Downloading the code

Once you have all necessary dependencies, you can download the scripts from our repository (either by cloning the repository or directly downloading the files above). Be sure to download all files in the "ExampleFiles" folder if you want to test the functionality of all the scripts with the commands provided below.

Running scripts on the example files

1. To run the `blockParse.py` script on a .fa file, you can run the following command: `python blockParse.py -f 3.fa`
   This produces a .fastq file (`3.fastq`) containing all identified probe sequences matching your provided criteria. To see additional command line arguments available for this script, you can run the python file with the `-h` argument (i.e. `python blockParse.py -h').
2. NGS alignment. For example, you can use Bowtie2 to align the newly generated set of candidate probes by running: `bowtie2 -x /path_to_hg38_index/hg38 -U 3.fastq --no-hd -t -k 100 --very-sensitive-local -S 3_u.sam`
   or `bowtie2 -x /path_to_hg38_index/hg38 -U 3.fastq --no-hd -t -k 2 --local -D 20 -R 3 -N 1 -L 20 -i C,4 --score-min G,1,4 -S 3.sam`
   ... where "path_to_hg38_index" is replaced with the path to the bowtie2 indices for your genome of interest. These commands produce .sam files (`3_u.sam` and `3.sam`) containing sequence alignment information, but require genome builds as described in the previous section. If you are just testing your scripts to make sure they are working properly, we have already provided the output `3_u.sam` and `3.sam` files in the example files directory for you to use to test subsequent scripts.
3. To process the .sam file produced by sequence alignment, use the `outputClean.py` script: `python outputClean.py -u -f 3_u.sam`
   or, optionally (requires sklearn for the LDA model, see above) `python outputClean.py -T 42 -f 3.sam`
   13 of 13 of the candidate probes should pass the first command (and 12 of 13 candidate probes should pass the specificity filtering with the 42C LDA model in the second command). To see additional command line arguments available for this script, you can run the python file with the `-h` argument (i.e. `python outputClean.py -h').
4. [Optional] Now, you can use `kmerFilter.py` to screen your probes against high abundance kmers (requires Jellyfish to be installed and in your path, and a Jellyfish dictionary, see instructions above). `python kmerFilter.py -f 3_probes.bed -m 18 -j 18 -j sp.jf -k 4`
   This command uses a Jellyfish dictionary containing information about high abundance kmers in the genome of interest to screen probes. (We have provided `sp.jf` as an example for you to test the python script, which should pass all 12 probes into the file `3_probes_18_4.bed` . However, you will need to generate your own Jellyfish dictionary for your desired genome in the real case!) To see additional command line arguments available for this script, you can run the python file with the `-h` argument (i.e. `python kmerFilter.py -h').
5. To convert your probe set to their reverse complements, you can use the `probeRC.py` script: `python probeRC.py -f 3_probes.bed`
   This creates a file, `3_probes_RC.bed` containing the reverse complements of all sequences in the original .bed file. To see additional command line arguments available for this script, you can run the python file with the `-h` argument (i.e. `python probeRC.py -h').
6. [Optiona] You can check for secondary structures of probes by calling NUPACK using the

`structureCheck.py` script: `python structureCheck.py -f 3_probes.bed -t 0.4`
 This command should pass 6 of 12 example candidate probes. Additional information can be seen in the produced `3_probes_sC.bed` file. To see additional command line arguments available for this script, you can run the python file with the `-h` argument (i.e. `python probeTm.py -h').
7. [Optional] To generate a list of melting temperatures for a given probe set, you can use the `probeTm.py` script: `python probeTm.py`
 or `python probeTm.py -f 3.txt`
 The first command will allow you to enter a sequence interactively to retrieve its computed melting temperature. The second command takes a two column .txt file with the sequence in column 2 (tab delimited) and outputs a new file (`3_tm.txt`) with a 3rd column of Tms.

That's all! If you made it through these all without any errors thrown about missing dependencies or modules, you are all set to run OligoMiner on your own computer. Happy FISHing!

Notes on running OligoMiner on new genomes

You'll need to download your genome of interest in FASTA format and prepare index/dictionary files for your NGS aligner and optionally Jellyfish. We recommend using unmasked files for dictionary file construction and repeat-masked files as the input files for `blockParse.py`

Contributing

We welcome commits from researchers who wish to improve our software. Please follow the git flow branching model. Make all changes to a topic branch off the branch `dev`. Merge the topic branch into `dev` first (preferably using `--no-ff`) and ensure everything works. Code will *only* merged into `master` for release builds. Hotfixes should be developed and tested in a separate branch off `master`, and a new release should be generated immediately after the hotfix is merged.

Questions?

Please reach out to Brian with any questions about installing and running the scripts.