MetaSearch Recipe

**0. Find a nice computer that you can use for a few days**

a. System requirements:

i. None of the software here requires massive memory or huge numbers of processors

ii. You will need sufficient disk space to handle the relevant files

If you have 1GByte available per 2GBase of sequence you should have no problem

This can be slightly minimized by downloading and converting files in groups

The minimum requirement will likely be 1GByte per 3-5GBase of sequence

iii. Memory requirements for downloading and searching are modest

8-16GB should be sufficient

sra-to-fasta conversion may be faster with more memory and/or use of fasterq-dump

iv. Mutiprocessor systems should run more quickly (we used an 8-core system)

v. Internet connection speeds can be limiting for the download step

An ethernet connection would be valuable but is not essential

vi. Disk access can be limiting for the search routines (which are otherwise very fast)

An SSD drive can speed this up greatly

If no SSD drive is available, but multiple conventional hard drives are available, you can

spread the fasta files among several drives and run multiple copies of the

Jazz18Heap program on each drive to speed up the process

SSD (either "SATA" or [better] M.2) hardware is much faster than conventional drive

Faster conventional drives (7200RPM) will be somewhat quicker

**1. Install SRA-tools (NCBI) on your system if not already installed**

a. Formal instructions on installing SRA Tools can be found at

https://github.com/ncbi/sra-tools/wiki/HowTo:-Binary-Installation

or here

https://ncbi.github.io/sra-tools/install\_config.html

b. There are a variety of discussions online of how to effectively use SRA Tools

Here is one (from Renesh Bedre [Texas A&M]-- with thanks)

https://reneshbedre.github.io/blog/fqutil.html

**2. Install IBM Aspera Connect on your system**

a. Instructions are (currently) accessible from these two addresses:

https://www.ibm.com/aspera/downloads/

https://www.ibm.com/support/knowledgecenter/SSXMX3\_3.9.8/kc/connect\_welcome.html

b. IBM seems to keep many older versions of the instructions on their website

You may need the most current to interface with the NCBI servers efficiently

**3. Obtain a list of the SRA datasets you would like to download.**

There are various means to do this, one is a follows:

Navigate a browser to the SRA search page at NCBI

https://www.ncbi.nlm.nih.gov/sra

Do a search to get a web list of results

A screenshot of a social media post

Description automatically generated

Your results might look like this

A screenshot of a cell phone

Description automatically generated

You can limit the files using the constraints on the left

(but be aware that categorization going into these classes may be inaccurate or incomplete)

Click on "Send Result to Run Selector". (There is also a note that the SRA data is now in the cloud. This will be wonderful when some documentation is available on how to use this.)

As of 2/17/20, the "new run selector" sometimes doesn't actually work (one common "fail" is shown below), but if this is a problem, just try again (this often takes several tries, with various semi-blank screens, lists of blank accession records, or a statement that nothing was matched despite the previous screen noting a number of query matches).

If there is still no success with the "New Run Selector" after a few tries, you can also click on "Revert to the old Run Selector"

Eventually you should get a list of runs with an option to download a metadata file.

A screenshot of a social media post

Description automatically generated This file is The list of run metadata will be given the name SraRunTable.txt, and is a comma-delimited text file with a header row. Trying the old interface gives a similar file that is tab delimited and with the run designations later in each row. Either should work with the downstream steps below.

**4. Fetch the individual SRA files**

The SRA-tools component that downloads information from the archive is called "prefetch". The program will download a copy of each requested dataset in a somewhat obscure format given a '.sra' file extension. You can skip this and directly download files in principal, but we found this to be highly variable for large sets of files with our internet connection. The python script "AllInOnePrefetcher" will organize and monitor the downloads as well as employing multiple threads for this process. A simple means to run the script is to place the SRARunTable.txt file in the /bin directory of the extracted SRA tools folder, and run the script

python AllInOnePrefetcher\_aa1\_021420.py ListFile='SRARunTable.txt'

This will download all of the indicated files into a directory specified by SRA-tools (generally /ncbi/sra). You can direct the files to a specified directory with

python AllInOnePrefetcher\_aa1\_021420.py ListFile ='SRARunTable.txt' destination=<mydir>

Running several threads will greatly speed the process and can be done with

python AllInOnePrefetcher\_aa1\_021420.py ListFile='SRARunTable.txt' **Threads=nn**

Ideally a number of threads that is twice the number of cores should give the most rapid overall run (e.g., Threads=16 for an 8-core system)

Here are instructions and options for *AllInOnePrefetcher* (also accessible with command line option '--help')

##

## -> AllInOnePrefetcher takes a list of NCBI-SRA accession numbers

## and downloads these to a specific folder

## -> Program operates with multple threads at once and tries continuously

## to download each requested file to avoid download failures

## -> Assumes that SRA tools and Aspera Connect are installed on your system

## -> Some files are protected by IRB and the program will generally skip these.

## To try to download, set irb=True (not tested)

## -> Very large files may be unwanted. Current Maximum download is set to 80G

## (larger files will be skipped), To change this, set MaxSize=##G

## -> Tested with Python 2.7 (will maybe run with Python 3 but not tested)

## -> Call Format

## python AllInOnePrefetcher <parameters/options>

## Required Parameters:

## ListFile=<SRARunTable\_file> \*You'll need to provide a list\*

## ListFile can be in any format as long as the first item is an SRA accession##

## Options:

## PrefetchProgram = <Path\_to\_Prefetch> \*Default is to look in current directory\*

## DestinationDirectory = <Destination\_Directory> \*Default is to use sra-tools setting\*

## Threads = <Number of Parallel threads to try> \*Default is 16\*

## MaxSize = <Maximum download size in Gigabytes> \*Default is 80\*

## IRB = <> \*Attempt to download IRB-protected files? [default is False]\*

## Delimiter = <> Column delimiter used in ListFile (generally tab or comma)

## Should autodetect for recent NCBI SRARunTable formats

## Otherwise can explicitly set with Delimiter=tab or Delimiter=comma

## RunColumn = <> Column # for RunID in ListFile.

## Should autodetect if you have retained the header row in the SRATable

## Defaults 0 comma-delimited (new RunTable fmt), 32 tab-delimited (old fmt)

## -> Copyright Fire Lab, Stanford University 021020, No Guarantees of any sort

##

**5. Extract the individual SRA files.**

You will need to extract the sequences with the SRA tools program "fastq-dump" (or if you can get the dependencies set up, "fasterq-dump"). You could do this with no custom scripts, just using a command line statement (e.g.

$ fastq-dump /ncbi/sra/\*.sra <other fastq-dump parameters>

The "AutomatedSRRToFastAConverter" script does some of the light lifting on this and arranges multi-core usage (particularly useful if attempts to use "fasterq-dump" meet the same challanges we have seen. The call syntax for AutomatedSRRToFastAConverter is

$ python AutomatedSRRToFastAConverter.py SourceDirectory=<mySourceDir> DestinationDirectory=<MyDestinationDir>

The only likely parameters to set with this are the location of the fastq-dump program (e.g.

fastqdump = ./sra-tools##/bin/fastq-dump

And Threads= (number of threads to use-- generally 2x cores will speed up the overall conversion effectively.) Threads=16 would be optimal for an 8 core system).

Here's instructions for AutomatedSRRToFastAConverter (also accessible with command line option '--help')

##

## -> AutomatedSRRToFastAConverter\_AC\_021720

## -> AutomatedSRRToFastAConverter takes a directory with NCBI-SRA files

## and converts all .sra files into fasta format using fastq-dump.

## Humble Note: You can also do this with

## a single command line argument ./fastq-dump /mydir/\*.sra

## but that may lose some flexibility in the call

##

## -> Program operates with multple threads at once and operates both from

## beginning to end and from end to beginning of a list of files

## so a single failure won't as easily stop the entire ship

##

## -> Assumes that SRA tools is installed on your system and that

## fastq-dump is available (fasterq-dump could also be used if you can get

## the fasterq-dump dependencies to work.

##

## -> Tested with Python 2.7 (will maybe run with Python 3 but not tested)

##

## -> Call Format

## python AutomatedSRRToFastAConverter\_ab\_021620 <parameters/options>

## Required Parameters:

## SourceDirectory = <Input directory> \*You'll need to provide\*

## DestinationDirectory = <Destination\_Directory> \*Default is to use current directory\*

## Options:

## FastQDumpProgram = <Path\_to\_fastq-dump> \*Default is to look in current directory\*

## or FasterQDumpProgram = <Path\_to\_fasterq-dump> \*Default is to look in current directory\*

## Threads = <Number of Parallel threads to try> \*Default is 16\*

## -> Copyright Fire Lab, Stanford University 021620, No Guarantees of any sort

##

## -> Version AC 021720 (First version with command line interface)

##

**6. Construct a fasta file that will serve as a search template.**

The file will have a ".fasta" structure

>SequenceName1

AGACCAGAGAGAGAGAGAGAGAGAGGAAGAGAG

>SequenceName2

GGACCAGAGAGAGAGAGAGAGAGAGGAAGAGAG

You can include as many sequences as you are interested in, but at some point the searching will slow down. This depends on memory and a few features of the system. Certainly a megabase of sequence should not particularly slow the search.

**7. Search through the metagenomic files for k-mers that match sequences in the input fasta file.**

Any match will be shown. The script used for this is Jazz18Heap (alternatives would be Blast, BWA, Bowtie, etc, but parameters for these programs would need to be adjusted [mostly using long word sizes] to maximize speed). Jazz18Heap is very quick when run using the optimized interpreter pypy and using multiple processors on a system. With a standard disk drive, the limiting factor may be speed of reading off the drive. With an SSD Drive (particularly with M.2 interface), the speed can be as high as 1GBase per second (or 10 million 100 base reads per second) on an 8-core system.

Calling Jazz18Heap can be done as follows

pypy Jazz18Heap\_AH\_02202020.py ReferenceFile=<MyReference.fasta> DataFiles=</path-to-fasta-gz-files>

Some examples

pypy Jazz18Heap\_AH\_02202020.py ReferenceFile=ViralSampler.fasta DataFiles=./\*.fasta.gz

A few important parameters for Jazz18Heap

DataFiles = <List of .fastq files, .fasta files, or NCBA-SRA accessions [e.g, SRR####]

Lists are comma delimited with no spaces in list

.fasta or .fastq files can be gzip compressed, although this somewhat slows the program down

.fasta and .fasta.gz data files must have exactly one sequence per line (no multiline sequences)

For .fasta files to be downloaded from NCBI-SRA archive this entails the command line parameter --fasta 0

\* Wildcards are allowed here as well, or list of files in a file with the extension .files

Providing a directory here will search all files in this directory or subdirectory for fasta and fastq data files

ReferenceFile = <FastA file with list of sequences to match k-mers from>

klen = <How long are the k-mers used>. Default klen = 32

Threads = <> An integer indicating how many separate processes to spawn (generally twice the number of cores, e.g. 16 for an 8-core system)

Full Instructions

**## Jazz18Heap.py version AH 02202020**

## Jazz18Heap a fast Python Script for finding matches to a probe sequence in large numbers of

## High throughput sequencing output files

## This program implrements a simple index search that will look for instances of sequence reads matching

## a reference in at least one k-mer.

## Jazz18Heap is intended to look for evidence of matches to a relatively short reference sequence

## e.g., less than 100KB, but probably workable up to several MB in a large number of high throughput sequencing experiments

## Jazz18Heap is intended for finding relatively rare sequence matches (not common matches)

## Jazz18Heap doesn't substitute for the many tools that align and track coverage.

## Jazz18Heap main goal is rapid identification of homologous sequences (homology defined as a perfect match to a long sequence-e.g. 32b for klen=32)

## Inputs are as follows (command line, Key=Value syntax)

## ReferenceFile = <FastA file with list of sequences to match k-mers from>

## ExcludeFile = <FastA file with sequences that will be excluded from matches>

## DataFiles = <List of .fastq files, .fasta files, or NCBA-SRA accessions [e.g, SRR####]

## Lists are comma delimited with no spaces in list

## .fasta or .fastq files can be gzip compressed, although this somewhat slows the program down

## .fasta and .fasta.gz files must have exactly one sequence per line (no multiline sequences)

## For .fasta files to be downloaded from NCBI-SRA archive this entails the command line parameter --fasta 0

## \* Wildcards are allowed here as well, or list of files in a file with the extension .files

## Providing a directory here will search all files in this directory or subdirectory for fasta and fastq data files

## Optional Parameters (will default to reasonable values if not set)

## OutFileBase = <Character String to Label Output Files with>

## ReportGranularity = <How many reads to process before reporting hit numbers (default is 1Million)>

## SearchGranularity = <How much distance between k-mers to be examined in each data read

## setting SearchGranularity=1 makes Jass18Heap look at every k-mer in every read

## setting SearchGranularity=8 makes Jass18Heap look at every 8th k-mer in every read

## higher numbers may miss a few hits but can greatly improve speed.

## setting to a large number (999999) ensures only one k-mer will be looked up per read

## SearchOffset = <Where to start jumping through each read for potential k-mers (zero means start at first base)

## fastqdump = <Where to look for fastq-dump -- this is only important if you are downloading files from NCBI SRA during execution of Jazz18Heap>

## klen = <How long are the k-mers used>. Default klen = 32

## snpAllow = <Set to True to allow a single mismatch in each k-mer (default is False)

## Circular = <Set to True to force every Reference sequence to be treated as a circle>

## Default : Uses the FastA name line-- if this line contains "Circular", the sequence is treated as a circle

## Multithreading: Jazz18Heap has the very primative multitasking ability to spawn a number

## of derivative processes for a large number of data files to be scanned. To use 16 Threads

## Set Threads=16 in the command line.

## Output:

## Output consists of a file-by-file list of hits, a log file with information on the run, and a FastA file with "Caught" Reeads

**## File-By-File Output Format**

## Column 1: Dataset\_ReadNumber (e.g. SRRXXX\_2 is the read 2 file from SRXXX)

## Column 2: Number of Reads extracted and analyzed

## Column 3: Number of bases analzyed

## Column 4: Number of hits from the viral sampler query

## Column 5: Total number of hit k-mers

## Column 6: Summary of hits. Each element has

## -> a query name (e.g. X51522.1\_phageP4)

## -> a position (e.g. p5012 starts at position 5012)

## -> a strand ('s' indicates a sense match starting at base , 'a' indicates an antisense match, and

## -> a k-mer match count (e.g. m4 indicates that four different k-mers were matched every nth k-mer being checked [n is the search granularity])

##

## FastA "Caught Read" files are a second output, with information about each read in the ID line

## ID Structure:

## DataFileName;LineNumberInDataFile;ReferenceFileName;ReferenceSequenceName;NumberOfMatchedKmers

## Running the program:

## Jazz18Heap only runs at full speed with a variant of Python (pypy) that included a just-in-time compiler

## Syntax Jazz18Heap RefFile=<MyRefFile> DataFiles=MyFastA1.fasta,MyFastA2.fasta.gz,MyFastQ\*.fastq <Other\_Parameters>

##

## Copyright 2020, Andrew Fire and Stanford University

## With thanks to Dae-Eun Jeong, Loren Hansen, Matt McCoy, Nimit Jain, Massa Shoura, Karen Artiles, Lamia Wahba, Nelson Hall