kvasirHGT Documentation

Release .6.8

Kevin Bonham, PhD

Feb 22, 2020

Contents:

Identification of horizontal gene transfer between sequenced microbial genomes

Kvasir is a Norse god associated with fermented beverages. According to Wikipedia,

> Extremely wise, Kvasir traveled far and wide, teaching and spreading knowledge. This continued until the dwarfs Fjalar and Galar killed Kvasir and drained him of his blood. The two mixed his blood with honey, resulting in the Mead of Poetry, a mead which imbues the drinker with skaldship and wisdom, and the spread of which eventually resulted in the introduction of poetry to mankind.

CHAPTER 1

Installation

The easiest way to use Kvasir is to install it from pypi:

```
$ pip install kvasirHGT
```

This should add the necessary scripts to your `PATH`.

You will also need install the other python Dependencies (biopython, pandas, pymongo), and mongodb

CHAPTER 2

Usage

2.1 Mongod

Kvasir uses a MongoDB database to store genome and HGT information, so you'll need to install mongodb and PyMongo before doing anything.

MongoDB will build a database and index on your local system, so you should make a folder to put them in. You'll never have to interact with the folder manually, so I just put it in a hidden folder in my home directory.

```
$ mkdir ~/.mongo_databases
```

Before you run any commands with kvasir, you need to launch a local MongoDB server using the `mongod` command:

```
$ mongod --dbpath ~/.mongo_databases
```

2.2 Adding Genomes

Kvasir can import genomes in Genbank format. You can pass either a single file, or a folder with many genbank files. Only files with extensions `.gb` or `.gbk` will be imported.

Each genome set should have a unique name - this will be the first argument for every script. For the following examples, I'll be looking at cheese genomes and the set will have the name `cheese1`

```
$ kv_import.py cheese1 -i path/to/genbank/files
```

I haven't yet built a way to check if you're trying to import the same genome multiple times, so use with caution! If you're not sure if you've already imported a file, you can query your MongoDB with the species name (see below for info on what is used as the species name). If you're not sure if you've imported *Awesomeus speciesus strain A*, launch python, and try the following:

```
import pymongo
db = pymongo.MongoClient()["cheese1"] # substitute "cheese1" for the name of your␣
↪genome set
db["genes"].find_one({"species":"Awesomeus speciesus strain A"})
```

If nothing is returned, then you haven't imported it yet. The text has to be exact though. If you're not sure, you can also get a list of all the species with any genes in the database:

```
db["genes"].distinct("species")
```

A note on Genbank formats

If your files aren't formatted correctly, you ~~may~~ will have issues. At one point, I tried to write in a way to accommodate all kinds of messed up formats, but it's just not worth it. So check your genomes with care.

In particular, be sure that each contig in the genome has a unique id (that's the first field in the `LOCUS` line). Kvasir uses this to determine which genes are located near each other, if there are duplicates, it may think that genes on separate contigs are next to each other. Each gene should also have a unique `locus_tag` field.

Most genbank files have a `SOURCE` identifier, which contains the genus, species and strain that a particular contig comes from. This is what kvasir will use to identify genes from the same species. If your genbank file doesn't have this, kvasir will use the name of the genbank file as the name of the species - but be careful! A single genbank file can contain contigs from multiple species - if your files are organized this way, make sure that they have a `SOURCE` field.

At some point, I'll write a convenience script to check through your genomes to make sure that they'll work. But until then - use with caution.

2.3 BLAST Searching

To make the next part work you need one more thing - the BLAST+ command line tools. This needs to be installed and accessible from your `$PATH`. In other words, you should be able to run `which blastn` and get a path spit back at you. If not - do some googling. I can't help you.

Assuming this is working (you've still got `mongod` running right?), it's time to start blasting. There are a couple of commands in the `blast.py` script. First, you've gotta make a blast database with all of the the genes in your database using `makedb`. This command will make 3 files, in the current directory by default, or in the directory you specify with `-b`

```
$ kv_blast.py cheese1 makedb -v -b ~/blastdbs
INFO:root:making BLAST database with protein coding sequences from cheese1 at /Users/
↪ksb/blastdbs
INFO:root:BLAST db created at /Users/ksb/blastdbs
```

Next, blastall will go through each species in your database and blast it against that database. Each hit will then be stored in your MongoDB with some metadata. This can take a long time - use the `-v` (verbose) flag to get status updates.

```
$ kv_blast.py cheese1 blastall -v -b ~/blastdbs
INFO:root:blasting Awesomeus speciesus strain A
INFO:root:Blasting all by all
INFO:root:Getting Blast Records
INFO:root:---> 500 blast records retrieved
...
```

In this case - I've put in a check so if you run this a second time, it will skip blasting any species that already have hits found in the database. You can use the `-f` flag to override this, but I wouldn't do that, because then you'll have a bunch

of duplicates. If you made a mistake and want to re-run it, I suggest clearing out the `blast_results` collection of your database. In python:

```
import pymongo
db = pymongo.MongoClient()["cheese1"] # substitute "cheese1" for the name of your␣
↪genome set
db["blast_results"].delete_many({"query_species":"Awesomeus speciesus strain A"})
```

. . . or if you want to remove all blast results and start over:

```
db["blast_results"].drop()
```

Eventually, I will add a tool so that you can just add a single species or a group of new species and blast them separately, but for now, you'll have to start from scratch each time.

2.4 Analysis

If your genome set contains only one example of any given genus, you're in luck! At this step, you can run the HGT analysis, which will look for all protein coding genes that are >99% identical, and put them into groups (see below). If you have groups of genomes that are in the same genus, however, it's worth verifying that they are not too closely related.

Dealing with closely related genomes

There are a few ways to approach this, and in kvasir they're unified under the idea of "species distance" and there are some useful functions `bin/kv_distance.py`.

If the genbank files you imported have a `SOURCE` line for the name of your organism, and it's formatted as `<genus> <species> <strain>`, you can use the script to calculate the distance based on Average Nucleotide Identity (ANI). This is calculated using a ruby script from enveomics. Measuring ANI using this method is only appropriate for closely related genomes, and as it stands, the script grabs all species in your MongoDB and calls the ANI script on pairs that have the same genus.

```
$ kv_distance.py cheese1 ani
```

This script pulls species from the MongoDB and then looks for all pairs of names to see if they're the same genus. It does that a bit naively, using `split()[0].split("_")[0]`, which will split the species name at spaces or _, and then take the first thing. So if you're looking at "Awesomeus speciesus strain A", this command will check if you have anything else with "Awesomeus" as the genus, like "Awesomeus_speciesus_strain_B", and will calculate the ANI between them and add them to a "species_distance" database in the MongoDB.

But be careful! If instead your species are "strain_1" and "strain_2", it will think they both have the genus "strain", whether or not they're related at all.

Once you've done this, you can export a distance matrix using these ANI calculations. Identical species will have an ANI distance = 0, and anything that is not the same genus will have a distance of 1. I highly recommend doing this to make sure the output is what you expect.

```
$ kv_distance.py cheese1 distance_matrix -o ~/Desktop/dm.csv
```

If ANI is not appropriate, or you have a different way to measure species distance (eg 16S similarity), you can make your own distance_matrix and import it. You have to make sure that the names of columns and rows are identical to the names from the `SOURCE` line of your genbank file (of if there's no `SOURCE` line, the file names). The table should have the structure:

	Awesomeus speciesus strain A	Awesomeus_speciesus_strain_B
Awesomeus speciesus strain A	0	0.1
Awesomeus_speciesus_strain_B	0.1	0

Assuming this is saved in a file `~/Desktop/ssu_dm.csv`, you can do:

```
$ kv_distance.py cheese1 distance_matrix -i ~/Desktop/ssu_dm.csv -t ssu
```

The `-t` parameter is the "distance type", which can be anything you want. The ANI script above uses `-t ani` by default. If you want to get your ssu distance matrix out at a different time, use:

```
$ kv_distance.py cheese1 distance_matrix -o ~/Desktop/returned_ssu_dm.csv -t ssu
```

Be sure your distance is actually a distance, and is between 0:1. So if two species have a 16S gene that is 85% identical, the distance should be 0.15.

In the next section, the default is to not use the species distance parameter or more precisely, the minimum distance (`min_dist`) parameter is set to 0. If you have closely related species, but don't set this, it will look like you have a lot more HGT than you actually do. In the paper, for ANI we used `min_dist = 0.11`.

Getting HGT groups

Now the fun stuff!

```
$ kv_analysis.py cheese1 groups -o ~/Desktop/groups.csv
```

This could take some time, depending on the number of species and how much HGT there is.

You can also tweak some parameters like minimum size of a protein coding sequence, distance between genes to be considered the same group, minimum identity to be considered HGT, and minimum species distance with various flags. Try using `kv_analysis.py -h` to see what options you have.

Eventually, there will be more options for analysis, but that's all for the moment.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search