The `quaff` Manual

Ian Holmes

October 30, 2015

Contents

1 Introduction

`quaff` is a pair HMM-based tool for aligning FASTQ reads to FASTA references, with the following features:

- it pre-filters and bands the DP algorithms by looking for diagonals with lots of k-mer matches

- you can train the pair HMM on sequences using the Forward-Backward algorithm
- you can align reads to references using the Viterbi algorithm
- you can also align reads to reads (i.e. look for overlaps), using Viterbi
- it attempts to model the FASTQ quality scores (using a negative binomial distribution), and also uses k-mer context for modeling substitution and/or gap-opening probabilities
- it's highly parallel: multithreaded, can run remote servers, or launch its own temporary Amazon EC2 cluster

The main usage modes of `quaff` are:

Alignment mode — align nanopore reads to a reference genome

Overlap mode — align nanopore reads to each other

Training mode — fit `quaff`'s underlying model parameters to data

Further information is available via the `quaff` usage message (which can be printed by "`quaff help`").

2 Installation

You should first download `quaff` from `https://github.com/ihh/quaff` or clone the `quaff` git repository

```
git clone https://github.com/ihh/quaff.git
```

Then unzip (if applicable) and `cd` into the `quaff` directory.
To build `quaff` you will need:

- EITHER of the following two compilers:
  - The clang C++11 compiler (version clang-700 or later)
  - The gcc compiler (version 4.8.3 or later), PLUS the Boost C++ library (version 1.53.0 or later)
- libz (compression library), version 1.2.8 or later
- libgsl (GNU Scientific Library), version 0.14.0 or later
- GNU make
- perl (only to run the tests)

Type:

- "`make aws-dep`" to use the `yum` package manager to download dependencies (if `yum` is installed on your system)
- "`make quaff`" to build in `bin/`
- "`make test`" to run tests
- "`make install`" to install in `/usr/local/bin/`

3 General features

3.1 File formats

Input sequence files can be provided in FASTA format, FASTQ format, or gzipped versions of those formats.

Model parameters are saved in a JSON format, as are expected emission and transition counts (computed by the Forward-Backward algorithm); the latter can be used as pseudocounts during subsequent training runs (effectively specifying a Dirichlet prior distribution on parameters). Alignments can be output in a variety of formats, including Stockholm, gapped FASTA, and SAM.

3.2 General features of the model

Several signals can be exploited in the pursuit of improved alignment accuracy. For sequencing technologies such as Oxford Nanopore, where the raw signal is derived from a convolution of overlapping k-mers, gap and substitution rates may depend on recent sequence context. An additional, and under-exploited, signal is available in the case of FASTQ data which contains quality scores, as well as base-called nucleotides. According to the usual definition of FASTQ format, these "Q-scores" are nominally to be interpreted as PHRED scores; i.e., the logarithm of the probability of error. Such an interpretation offers one way to incorporate Q-scores in a statistical analysis. However, the distributions of Q-scores from different sequencing technologies and base-calling pipelines may be very different and there can be no guarantee that the relationship between Q-score and actual observed error rate is consistent across such different regimes.

Pair Hidden Markov models (HMMs), along with transducers (their input-conditioned relatives), are well-suited to the problem of aligning noisy, long reads. The emission and transition parameters of an HMM can be learned in an unsupervised way (i.e. without user-supplied training alignments), using the Baum-Welch algorithm, a version of the Expectation Maximization (EM) algorithm. The expected emission and transition counts computed by EM are themselves relevant to some statistics of interest (for example, sequence coverage, percentage identity, or indel frequencies). Furthermore, HMMs (and transducers) are easily extended. Incorporating adjunct data into the HMM emissions (such as quality scores) is straightforward; and transducer theory allows the systematic derivation of elaborated alignment tasks (e.g. finding the

overlap of two reads, hypothesizing that they were both derived from a single unknown reference) without the need to re-parameterize the elaborated model.

`quaff`'s underlying transducer incorporates the k-mer signal directly by allowing the substitution and gap probabilities at any given position to depend on the last k nucleotides of the read sequence. (It would be a better reflection of reality to allow the weights to depend on the last k nucleotides of the reference sequence; however, since the reference is not directly observed in many applications, such as read-to-read alignment, this would increase the computational expense of those applications.)

Quality scores are modeled using a negative binomial distribution with emission-dependent parameters. They are required for training, but are optional in alignment and overlap modes.

3.3 General options

General options that apply to all usage modes:

Option	*Effect*
`-params F`	Load parameters from file `F`
`-ref F`	Load reference sequences from file `F`, which may be in FASTA, FASTQ, or zipped/gzipped versions of those formats. (Quality scores in FASTQ files will be ignored for reference sequences.) You can specify more than one reference sequence file.
`-read F`	Load reads from file `F`, which may be in FASTA, FASTQ, or zipped/gzipped versions of those formats. You can specify more than one reference sequence file.
`-fwdstrand`	The default behavior is to consider each reference sequence twice: once as loaded, once reverse-complemented. This option turns off the reverse complement.
`-global`	The default behavior is to do alignment that is global in the read, local in the reference sequence. This option makes it global in both.
`-null F`	Load a null model from file F (default is to compute it automatically from the read sequences)
`-savenull F`	After computing the null model, or loading it, save it to file F

3.4 Word-matching

To accelerate the dynamic programming (DP) algorithms, `quaff` uses a strategy of pre-filtering by word-matching, as used by programs such as BLAST and DALIGNER from the Myersphere. Specifically, the DP is constrained to fixed-size diagonal bands around a small number of seed diagonals that contain more than a certain threshold of matching k-mers between the sequences being aligned. These thresholds can be specified directly by the user; alternatively, `quaff` can be configured to select the largest threshold that will fit either within a user-specified memory limit, or within available system memory.

Option	*Effect*
`-kmatch k`	Specify length of k-mers for pre-filtering heuristic
`-kmatchn N`	Specify threshold number of k-mers that must match on a given diagonal, before that diagonal is used to seed a band through the DP matrix
`-kmatchband W`	Specify width of a band through the DP matrix
`-kmatchmb M`	As an alternative to `-kmatchn`, set the threshold just high enough that the banded DP matrix fits within `M` megabytes of memory
`-kmatchmax`	As an alternative to `-kmatchmb`, set the threshold just high enough that the banded DP matrix fits within available memory, divided by the number of threads that `quaff` is using
`-kmatchoff`	Turn off the word-matching heuristic, do full DP. Typically this is very slow and should be avoided except perhaps for small test datasets

3.5 Logging

The default is silent operation. Use `-v` to turn on logging, `-vv` or `-v2` to make it a bit more verbose, and so on (`-vvv` is equivalent to `-v3`, `-vvvv` is equivalent to `-v4`, etc.)

Any problems can often be diagnosed by turning up the logging to `-v5` or so.

As a debugging feature, you can also turn on all logging within specific functions or source files with `-log FUNCTION_NAME` or `-log SOURCE_FILE_NAME`, respectively.

The log is printed on standard error. By default, log messages are colorized by log level, using ANSI terminal color codes. Piping through `less -r` preserves these color codes (just using regular `less` without options may fill your output with nonsense characters). Alternatively, they can be turned off using the `-nocolor` option.

4 Usage modes

The main usage modes of `quaff` are:

Alignment mode — align nanopore reads to a reference genome

Overlap mode — align nanopore reads to each other

Training mode — fit `quaff`'s underlying model parameters to data

The algorithms used by alignment, overlap and training are (essentially) versions of Viterbi and Baum-Welch. It should be noted that the maximum-likelihood alignment, as returned by the Viterbi algorithm, is not necessarily the most accurate alignment obtainable with a given parameterization: the decision-theoretic "optimal accuracy" alignment is better. Nor indeed does the maximum-likelihood parameterization, as given by Baum-Welch, in general yield

the most accurate Viterbi alignments of any parameterization. Nevertheless, Quaff implements Viterbi alignment (because it is around threefold faster than "optimal accuracy") and Baum-Welch training (because, to our knowledge, no unsupervised training algorithm yields more accurate parameterizations).

4.1 Alignment mode

```
quaff align refs.fasta reads.fastq >align.stockholm
```

The alignment algorithm is Viterbi on a pair HMM.
Alignment options:

Option	*Effect*
`-threshold`	Specifies the score threshold for printing an alignment. The default is zero, so only positive-scoring alignments will be printed (it is a log-odds score, so a positive score means the likelihood of an alignment is higher than the likelihood under the null model.)
`-nothreshold`	Don't use any score threshold, just print all alignments (or best alignment for every read, depending on `-printall`)
`-printall`	Prints every reference-read alignment that is above the scoring threshold, as opposed to just printing the best reference alignment for each read (which is the default).
`-noquals`	Ignore the quality scores when doing an alignment; treat them as missing data, just do a DNA-DNA alignment
`-format F`	Specify the output format as `F`, which should be one of `fasta`, `stockholm`, `sam` or `refseq`. The latter (`refseq`) just prints the part of the reference sequence that matches the read.

4.2 Overlap mode

```
quaff overlap reads.fastq
```

The read-overlap algorithm is a variant of Viterbi that probabilistically sums degenerate paths through gap states (so that biologically unmeaningful permutations of the gap order are marginalized). The underlying pair HMM is a minimal (3-state) approximation to a larger (27-state) pair HMM derived using phylocomposer, which implements algorithms for combining transducers (specifically, the 27-state machine arises from the intersection of two reference-to-read transducers, composed with a single-state generative HMM for the common underlying reference). The 3-state approximation trades meticulous accuracy for pragmatic run times.

The same options apply as in Alignment mode.

4.3 Training mode

```
quaff train refs.fasta reads.fastq >params.json
```

Training uses the Baum-Welch (EM) algorithm. Various options can be used to control the EM convergence, and other aspects of training:

Option	*Effect*
`-maxiter N`	Specify maximum number of iterations before EM will be stopped
`-mininc F`	Specify the minimum *fractional* increase in the log-likelihood before EM is considered to have (approximately) converged, and is stopped
`-maxreadmb N`	A crude way of down-sampling the training set by using only the first `N` megabytes of the reads.
`-force`	By default, reads that are more likely to have been generated by the null model (vs being derived from a reference sequence) are not used in training, because they are probably nonsense. This option forces them to be used anyway

Other options control the model parameters and prior:

Option	*Effect*
`-saveparams F`	By default the trained parameters are printed to standard output. This option saves them to a file `F` (and does so after each iteration of the EM algorithm, so you can get some idea of progress during long runs).
`-savecounts F`	The E-step of the EM algorithm involves computing expected counts for each type of substitution and insertion-deletion event (corresponding to emission and transition events in the pair HMM). These counts may be of interest in themselves, and furthermore, they can be used as "pseudocounts" (i.e. the hyperparameters of a Dirichlet prior) to guide subsequent training runs. The `-savecounts` option saves these counts to a file `F` for later usage or analysis.
`-savecountswithprior F`	Like `-savecounts`, but adds in the Dirichlet prior pseudocounts before saving.
`-prior F`	Loads Dirichlet pseudocounts from a prior file, which may have been created with `-savecounts`, `-savecountswithprior` or `-saveprior`.
`-saveprior F`	Saves a copy of the Dirichlet pseudocounts to a file

Yet more options control the structure of the underlying pair HMM itself:

Option	*Effect*
`-suborder K`	Make the substitution probability parameters depend on the previous `K` read nucleotides
`-gaporder K`	Make the indel probability parameters depend on the previous `K` read nucleotides
`-order K`	A shorthand for `-suborder K -gaporder K`

5 Parallel operation

For additional speed, `quaff` offers several multiprocessor modes to parallelize computations on large read datasets:

1. It can utilize multi-core architectures on a single machine by distributing DP jobs over a thread pool.

2. It can launch server jobs on remote machines, effectively increasing the size of the thread pool by using an existing cluster. These server jobs are long-running, and are controlled by the master node over sockets. If the Amazon Web Services (AWS) command-line tools are installed (and the user has an AWS account), `quaff` can instantiate a temporary cluster on the AWS Elastic Compute Cloud (EC2), download and build itself on the temporarily created EC2 instances, and use them as servers for the duration of the run (automatically terminating the instances after the run). In order to distribute data to server nodes, `quaff` can either transfer files using the `rsync` command, or can alternatively use 'buckets' of the AWS Simple Storage Service (S3). None of these parallel functions require any queueing or other cluster management software, except `ssh`, `rsync` and (in the case of EC2/S3) the AWS command-line tools.

3. If a job queueing system (such as Portable Batch System or Sun Grid Engine) is installed, together with NFS, then `quaff` can use these to distribute the workload over a cluster.

Parallelization is via a thread pool, which can be extended over a cluster. If IP addresses (or AWS credentials) are given, then jobs are run over sockets, and (if NFS is unavailable) files may be synchronized using S3 or `rsync`. Alternatively, jobs can be run using a queueing system (such as PBS or SGE), in which case NFS is required for both job synchronization and file sharing.

By default, `quaff` assumes all data files are in the same place on the server. You can copy them across using `-rsync`, or `-s3bucket`, or other means (eg NFS).

5.1 Running in multiple threads

Use the `-threads N` option to run a thread pool of `N` parallel compute threads, or `-maxthreads` to automatically detect the number of cores on the machine and use that many threads. **WARNING:** do NOT do this unless you are in full control of the machine. For example, if you are running `quaff` on a cluster, avoid this option unless you are sure that the cluster software is not already loading multi-core worker nodes with multiple jobs!

5.2 Running on a cluster

If you have a cluster and a list of hostnames (or IP addresses), with `ssh` access into each machine, you can have `quaff` fire up a bunch of worker nodes, effectively extending the thread pool across the cluster.

Typically you will need to do several things to make this work:

- Tell `quaff` where to find your `ssh` private key

- Tell `quaff` the address of the machine you want to run a worker on, the number of worker threads you want to run on that machine, and the port range on which the worker threads should listen (all three of these are specified with the `-remote` option)
- Tell `quaff` the location of the `quaff` binary on the remote machine(s)
- Ensure that data files are synchronized. There are several ways of doing this: NFS, `rsync`, or Amazon S3 buckets. The first (NFS) should be seamless without `quaff`; the latter two (`rsync`/buckets) require some complicity by `quaff` to make them seamless.

The `-remote` option has various different forms:

Option	*Effect*
`-remote H`	Uses `ssh` to launch a single-threaded `quaff` worker on host `H`, listening at the default port (8000)
`-remote U@H`	Uses `ssh` (with username `U`) to launch a single-threaded `quaff` worker on host `H`, listening at the default port (8000)
`-remote H:P`	Uses `ssh` to launch a single-threaded `quaff` worker on host `H`, listening at port `P`
`-remote U@H:P`	Uses `ssh` (with username `U`) to launch a single-threaded `quaff` worker on host `H`, listening at port `P`
`-remote H:MIN-MAX`	Uses `ssh` to launch a multi-threaded `quaff` worker on `H`, with one thread listening on each port in the range `MIN` to `MAX`
`-remote U@H:MIN-MAX`	Uses `ssh` (with username `U`) to launch a multi-threaded `quaff` worker on host `H`, with one thread listening on each port in the range `MIN` to `MAX`

On the client, `quaff` opens one socket per remote thread (plus one ssh job per server), so you may need to raise the system limits on the number of files/sockets per process (e.g. OSX 10.10 limits you to 128 sockets/process by default).

Unless you explicitly specify the `-threads` option, running remote workers will turn off all local processing (i.e. implicitly setting `-threads 0`).

Other options for using remote servers include

Option	*Effect*
`-sshkey FILE`	Specify location of `ssh` private key file
`-sshpath PATH`	Specify path to `ssh` executable
`-remotepath PATH`	Specify path to `quaff` executable on remote machines (must be in the same place on all remote servers)

5.3 Running on AWS Elastic Compute Cloud (EC2)

`quaff` can launch a temporary Amazon cluster if you have the `aws` command-line interface tools. Relevant options include:

Option	*Effect*
`-ec2instances N`	Launch `N` temporary EC2 instances as servers, shutting them down after computation has finished. This is the main option you need to use EC2, but you will typically need a few others as well (see below; many of these options have sensible defaults, but `-ec2key` is essential, and `-ec2group` is advised).
`-ec2ami AMI`	Specify the Amazon Machine Image to use. `quaff` will download itself (and pre-reqs), and build/install itself, on this image, so a generic Amazon Linux image should suffice. However, the AMI you choose must be available in your EC2 region. If none is specified, `quaff` will attempt to use a sensible default.
`-ec2type TYPE`	Specify what EC2 instance type you want to use (`m3.medium`, `c3.large`, etc.). The default is `m3.medium`.
`-ec2cores N`	You must also specify the number of cores that your instance type has, so that `quaff` knows how many worker threads to start on it. The default is 1.
`-ec2user USER`	This is the user-name to log into the instance. Default is `ec2-user`
`-ec2key KEYPAIR`	Name of the key-pair you want to use to log in. This must match the `ssh` key you specify with `-sshkey`
`-ec2port MINPORT`	The port that the worker thread will listen on (or the first port in the range, if using a multithreaded server, i.e. if your `-ec2cores` specifies more than one core). This must be compatible with the security access rules implied by your `-ec2group`. Default is 8000
`-ec2group GROUP`	Name of the security group you want AWS to use. This security group should allow incoming connections on port 22 (`ssh`) and on the ports you want worker threads to listen on (i.e. the `N` ports starting at `MINPORT`, where `N` and `MINPORT` are specified by `-ec2port MINPORT -ec2cores N`)

AWS requires a few fiddly things to work correctly, and clusters can take a while to launch, making debugging tricky. A few things to check: ensure that your AWS credentials are set (i.e. `AWS_ACCESS_KEY_ID` & `AWS_SECRET_ACCESS_KEY` environment variables) and that the shell you are using passes these variables to `quaff`. You must use an AMI consistent with your `AWS_DEFAULT_REGION` (the default AMI is a standard Amazon EC2 Linux for us-east-1). A standard AMI should be fine: `quaff` downloads prereqs and builds itself.

Also ensure that the specified AWS security group allows incoming connections on ports 22 (ssh) and on ports $N \ldots N + C - 1$ where N was the lowest port specified on the command-line and C is the number of core.

Finally, you will need to synchronize files. This is described in more detail in the sections below.

An example of how to use EC2:

```
quaff train
genomes/bacteria/ecoli.fasta data/nanopore/reads.fastq.gz
-ec2instances 10 -ec2type c3.2xlarge -ec2cores 8
-ec2key MYKEY -sshkey MYKEY.pem
-s3bucket MYBUCKET
-v5 -kmatchmb 50
-savecounts nanopore-counts.json
-saveparams trained-params.json
```

If you can't get AWS to work, try turning on verbose logging (e.g. `-v5`).

WARNING: `quaff` makes every effort to clean up rogue EC2 instances (e.g. catching aborts and interrupts), but please check! No liability will be accepted by the `quaff` developers for running up large EC2 bills due to zombie instances! You're on your own, mate.

5.4 Synchronizing files via rsync

One way to keep files synchronized with remote servers is to use the `-rsync` option. This will use the `rsync` utility (over `ssh`) to copy the relevant data files into a staging directory on the server. (The staging directory is a temporary directory automatically created under the path `/tmp/quaff`.)

`quaff` will automatically strip off the path information from the filenames that you give it and replace them with the path to the staging directory when passing filenames to the server, so you shouldn't need to worry about it getting confused in that way, although that does not rule out the possibility that it may get confused in other ways (e.g. if you have lots of different read files with the same filenames in different subdirectories).

If `rsync` is on a nonstandard place on your system then you can use `-rsyncpath` to tell `quaff` where it is.

5.5 Synchronizing files via AWS Simple Storage Service (S3)

A potentially more efficient way to synchronize files, especially if you are running worker jobs on EC2 (or running repeated `quaff` jobs using the same data files), is to synchronize via an Amazon S3 bucket. Use `-s3bucket BUCKETNAME` to tell `quaff` what bucket to use.

Note that you will also need to have the AWS CLI tools installed and your AWS credentials set via environment variables, as is the case when running jobs on EC2.

5.6 Running on a queueing system

Yet another way of parallelizing `quaff` is to use a cluster management system such as Portable Batch System (PBS) or Sun Grid Engine (SGE).

`quaff` uses a minimal interface to systems like this, which should increase the chances that it will work seamlessly on a broad range of such systems (one can dream). The only requirements are:

1. The system should have a utility like `qsub` (so-called on PBS and SGE) for submitting a job to the queue.
2. All nodes on the queue should be running NFS, and the user should have write-access to an NFS directory (this is used for sharing data files between nodes, and also for synchronizing jobs without relying on the queuing system's syntax for doing that, which can vary widely).

Relevant options:

Option	*Effect*
`-qsubjobs N`	Specify maximum number of jobs that will be simultaneously submitted to the queue at any one time. This option MUST be nonzero to use the queueing system. The reason `quaff` does not just submit all jobs at once? Well, mostly because queueing uses the same (fixed-size) thread pool mechanism as the other parallel options. However, this also has side benefits, e.g. it avoids filling up the queue with thousands of jobs, or filling up the NFS with thousands of temporary files.
`-qsub PATH`	Specify the path to the executable that is used to submit jobs (default is `qsub`). One way to test that queueing is set up correctly is to set this option to `/bin/sh`, which will just execute the job synchronously.
`-qsubopts OPTS`	Specify options to the `qsub` program. This is typically where you would specify what queue you want to use.
`-qsubheader FILE`	Specify a header file for the temporary shell script file that is created and passed to `qsub`. By default the header is just the shebang line, `#!/bin/sh`. However, some queueing systems may require extra information here, e.g. environment variables, or (as with PBS) additional configuration directives that are specified as comments in the script file.

Since the queueing code requires NFS to work, it cannot be used with `rsync` or S3-based file synchronization.