
Tentacle Documentation

Release 0.1.1b

Fredrik Boulund, Anders Sjögren, Erik Kristiansson

Oct 09, 2017

Contents

1	Citing Tentacle	3
2	Documentation:	5
2.1	Introduction	5
2.2	Download	7
2.3	Installing Tentacle	8
2.4	Tutorial	11
2.5	Running Tentacle	17
2.6	Tentacle output	19
2.7	Parsers	19
2.8	Coverage	20
2.9	Customizing modules in Tentacle	21
2.10	Choosing mapper	21
2.11	Performance evaluation	23
2.12	Indices and tables	26

Authors Fredrik Boulund, Anders Sjögren, Erik Kristiansson

Contact fredrik.boulund@chalmers.se

License GPL

Welcome! This is the documentation for Tentacle 0.1.1b, last updated Oct 09, 2017.

The documentation is available online at

<http://bioinformatics.math.chalmers.se/tentacle/>.

Tentacle is published as open-source under the GNU Public License (v3) and you are welcome to look at, improve, or come with suggestions for improvement to the code at the project's [Bitbucket](#) page.

CHAPTER 1

Citing Tentacle

If you find Tentacle useful and use it in your research, please cite us!. The framework is described in the following paper:

Boulund, F., Sjögren, A., Kristiansson, E. (2015). Tentacle: distributed gene quantification in metagenomes. *Giga-Science*, **4**:40, DOI: [10.1186/s13742-015-0078-1](https://doi.org/10.1186/s13742-015-0078-1)

Introduction

Overview

The Tentacle framework is developed to enable researchers to leverage high-performance computer (HPC) systems to quantify genes in large metagenomic data sets.

Tentacle provides a way to distribute resource intensive mapping tasks, such as mapping large numbers of metagenomic samples to a reference, in order to reduce the time required to analyze large data sets. Tentacle makes it possible to define what mapper and mapping criteria to use, and computes the coverage of all annotated areas of the reference sequence(s).

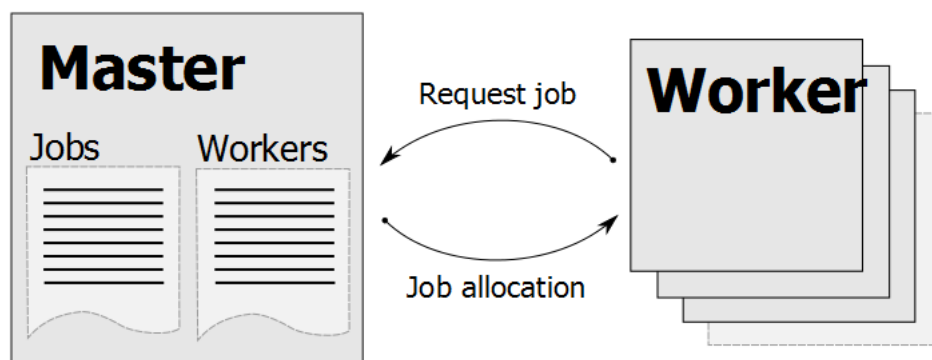


Fig. 2.1: Tentacle uses a master-worker scheme that has Master process that maintains a list of mapping jobs that should be run. The Worker processes (normally running on computer nodes in a cluster) connects to the Master process and asks for a mapping job to run.

More information available on how Tentacle computes coverage is available in section [Coverage](#). Further information on the implementation will also be published.

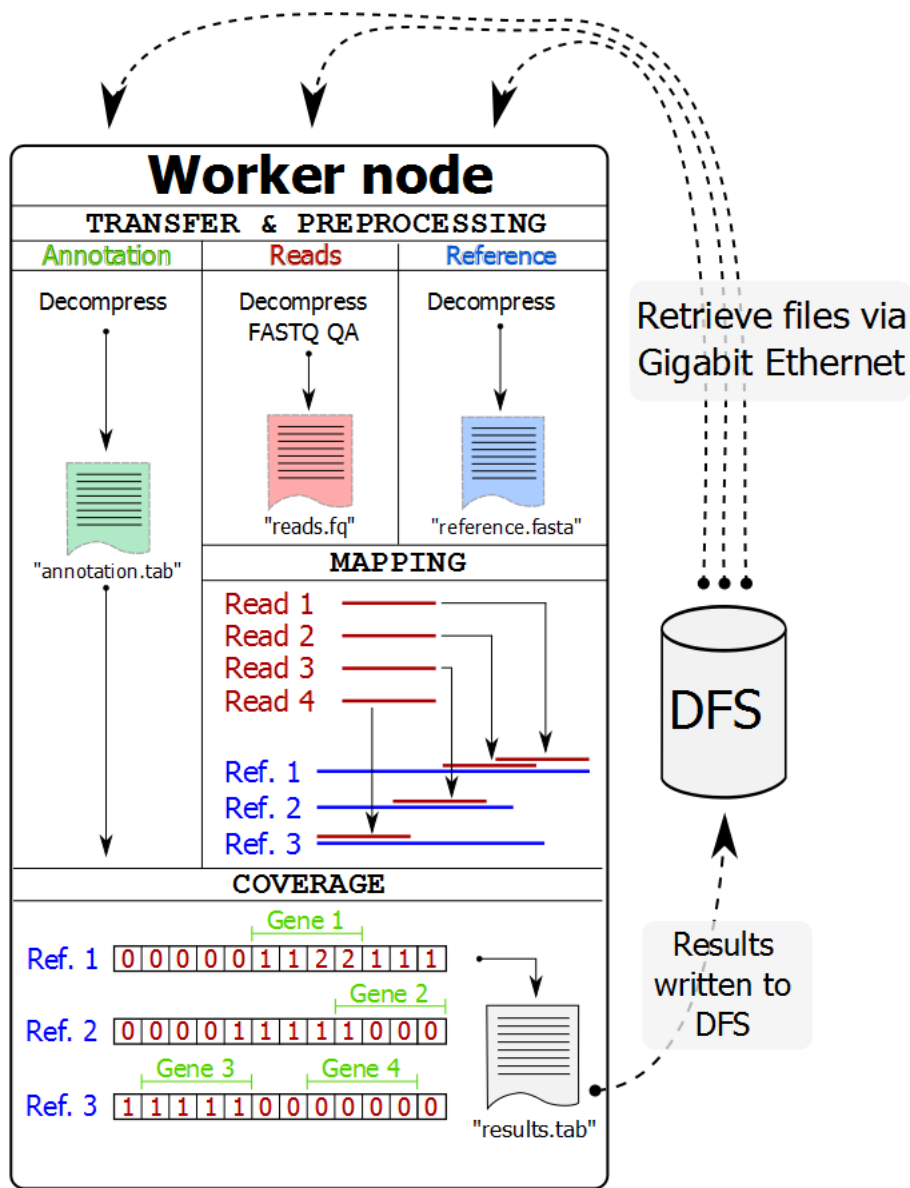


Fig. 2.2: The Worker processes (normally on a separate computer node in a cluster) perform all their work without interaction with the Master process. They retrieve the files via network from a distributed file system (DFS), perform preprocessing, mapping, and coverage calculations and then write the results back to the distributed file system. After completing a job, the Worker process requests a new one from the Master process.

Use examples

Tentacle is well suited for the following tasks:

- Map samples of metagenomic reads back to previously annotated contigs to determine the presence and abundance of certain genes. Check out [TUTORIAL 1. Mapping reads to contigs \(pBLAT\)](#).
- Map metagenomic reads to a reference database of e.g. antibiotic resistance genes. Check out [TUTORIAL 2. Mapping nucleotide reads to amino acid database \(USEARCH\)](#).

Abstract

Metagenomics is the study of microorganisms by sequencing of random DNA fragments from microbial communities. Since no cultivation of individual organisms is required, metagenomics can be used to analyze the large proportion of microorganisms that are hard or impossible to grow in laboratories. Metagenomics therefore holds great promise for understanding complex microbial communities and their interactions with their respective environments. The analysis of metagenomes in human gut, oral cavities and on our skin can, for example, provide information about what microorganisms are present and their effects on human health. The introduction of high-throughput DNA sequencing has significantly increased the size of the metagenomes which today can contain trillions of nucleotides (terabases) from one single sample. Current methods are however not designed with these large amounts of data in mind and are consequently forced to significantly reduce the sensitivity to achieve acceptable computational times. We have therefore developed a new method for distributed gene quantification in metagenomes. In contrast to many existing methods that are designed for single-computer systems our methods can be run on computer clusters and grids. Through efficient data dissemination and state-of-the-art sequence alignment algorithms the framework can rapidly and accurately estimate the gene abundance in very large metagenomes while still maintaining a high sensitivity. The output of our framework is adapted for further statistical analysis, for example comparative metagenomics to identify both quantitative and qualitative differences between metagenomes. Our method is shown to scale well with the number of available compute nodes and provides a flexible way to optimally utilize the available compute resources. It provides fast and sensitive analysis of terabase size metagenomes and thus enables analysis of studies of microbial communities at a resolution and sensitivity previously not feasible.

Copyright and license

Tentacle was developed at Chalmers University of Technology and is Copyright Fredrik Boulund, Anders Sjögren, and Erik Kristiansson. The code is freely available for use, re-use, and modification under the GPL (see [Citing Tentacle](#)).

Download

Download Tentacle

There are several ways to acquire Tentacle. The easiest is to just download and install Tentacle directly from the [Bitbucket repository](#):

```
(tentacle_env)$ pip install -e hg+http://bitbucket.org/chalmersmathbioinformatics/
↪tentacle/#egg=Tentacle
```

This process is further described in [Download and install Tentacle](#). If this for some reason does not work for you, another possibility is to download a release tarball from our servers and install it using pip:

```
(tentacle_env)$ wget http://bioinformatics.math.chalmers.se/tentacle/download/
↪tentacle-0.1.0b.tar.gz
(tentacle_env)$ pip install tentacle-0.1.0b.tar.gz
```

The third option is to download it directly from the [Bitbucket repository](#) using your browser of choice.

As a last resort if everything else fails, the entire [Bitbucket repository](#) can be cloned and used to install Tentacle without downloading a compressed archive. Clone the repository using Mercurial (hg) like this:

```
hg clone http://hg@bitbucket.org/chalmersmathbioinformatics/tentacle tentacle
```

You can then follow the instructions in [Download and install Tentacle](#) to install Tentacle from the repository clone.

Tutorial data

To follow the tutorial you need the data referred to by the tutorial instructions. It can be downloaded from here:

http://bioinformatics.math.chalmers.se/tentacle/download/tentacle_tutorial.tar.gz

Installing Tentacle

Here follows instructions on how to setup the environment required for Tentacle. A note on pathnames: The directory in which Tentacle is installed will be referred to as `$TENTACLE_ROOT` for convenience. Also, aside from in the section on how to install, prepare, and activate virtualenv, the `(tentacle_env)` prefix to the command line will be omitted for brevity. Note however, that all commands that utilize the Tentacle framework will require the virtualenv to be activated.

Dependencies

Tentacle depends on a number of Python packages. If they are not installed they will automatically be installed if you install Tentacle using `pip`. Should you decide to run Tentacle without installing it, you need to make sure that the required packages are available in your local Python installation. Below is a list of the required Python packages (tested versions in parenthesis).

- cloud (2.8.5)
- gevent (1.0)
- greenlet (0.4.2)
- msgpack-python (0.4.1)
- numpy (1.8.2)
- psutil (1.2.1)
- pyzmq (13.1.0)
- zerorpc (0.4.4)

Non-Python dependencies

To run Tentacle several non-Python programs are required as well. They should be installed and available on your `$PATH` or put in `$TENTACLE_ROOT/dependencies/bin/<system>`, where `<system>` is either Linux or Darwin depending on if you run Linux or OS X. The required software is listed below (tested version in parenthesis):

- seqtk (1.0-r45). Used for high-performance FASTQ to FASTA conversion. <https://github.com/lh3/seqtk>

Some software is only required if you required the functionality offered by them:

- FASTX toolkit (0.0.13): `fastq_quality_filter`, `fastq_quality_trimmer`. http://hannonlab.cshl.edu/fastx_toolkit/

Note that a mapper (sequence alignment software) is also required. See section [mappers](#) for information about what mappers are supported.

Python virtualenv

Python 2.7

Since Tentacle requires Python 2.7, make sure that this is the Python version used when creating your virtualenv. It is possible to specify a Python binary for use in the created virtualenv with the `--python` argument to `virtualenv`. E.g. `virtualenv --python=/usr/bin/python2.7 tentacle_env`.

The recommended way to use Tentacle is to setup a Python virtualenv (virtual environment), into which all required packages are installed. You can read more about virtualenv on their [official website](#).

To install virtualenv on your local computer cluster might require administrator privileges, contact your server administrator or support if it is not already available on your system.

Assuming that virtualenv is installed and globally available on your system, create a virtual environment in which to run Tentacle:

```
$ virtualenv tentacle_env
```

Activate the newly created (and empty) virtualenv by running the activate script (this will change your prompt to show that the environment has been activated, as illustrated below):

```
$ source tentacle_env/bin/activate
(tentacle_env)$
```

Download and install Tentacle

The Tentacle programs and framework is distributed via its [Bitbucket](#) repository. This requires that the distributed version control system [Mercurial](#) is installed. The most recent version can be automatically downloaded and installed directly from the online repository with the following command:

```
(tentacle_env)$ pip install -e hg+http://bitbucket.org/chalmersmathbioinformatics/
↪tentacle/#egg=Tentacle
```

Running this command will install Tentacle along with all dependencies (it will download any required dependencies that might be missing). Take a look at the output to make sure all packages install correctly into the virtualenv. Note that some of the required packages might in turn have further dependencies, these will install automatically. You can verify that all packages installed correctly by running `pip list` to see a listing of all packages that are installed in your virtualenv.

The `pip` installation will automatically make links to the three program files `tentacle_local.py`, `tentacle_slurm.py`, and `tentacle_query_jobs.py` into the `bin` directory of the virtualenv. You can call these from the command line to launch Tentacle from any folder.

Download and installing Tentacle manually

If [Mercurial](#) is unavailable or you require to install Tentacle without downloading it from the [Bitbucket repository](#), Tentacle can also be installed from a downloaded compressed archive. However, to get the most recent version we recommend installing it directly from the [Bitbucket repository](#). If you are unable to install Tentacle from the repository directly, you can install it using a release tarball as described in [Download](#).

The Tentacle programs and framework is distributed via its **‘Bitbucket repository’_** repository. This requires that the distributed version control system [Mercurial](#) is installed. The most recent version can be automatically downloaded and installed directly from the online repository with the following command:

```
(tentacle_env)$ pip install -e hg+http://bitbucket.org/chalmersmathbioinformatics/  
↪tentacle/#egg=Tentacle
```

Running this command will install Tentacle along with all dependencies (it will download any required dependencies that might be missing). Take a look at the output to make sure all packages install correctly into the virtualenv. Note that some of the required packages might in turn have further dependencies, these will install automatically. You can verify that all packages installed correctly by running `pip list` to see a listing of all packages that are installed in your virtualenv.

The `pip` installation will automatically make links to the three program files `tentacle_local.py`, `tentacle_slurm.py`, and `tentacle_query_jobs.py` into the `bin` directory of the virtualenv. You can call these from the command line to launch Tentacle from any folder.

Downloading and installing Tentacle manually

If [Mercurial](#) is unavailable or you require to install Tentacle without downloading it from the **‘Bitbucket repository’_**, Tentacle can also be installed from a downloaded compressed archive. However, to get the most recent version we recommend installing it directly from the **‘Bitbucket repository’_**. If you are unable to install Tentacle from the repository directly, you can install it using a release tarball as described in [Download](#).

To install Tentacle from a downloaded compressed archive into the virtualenv, make sure the virtualenv is activated and use `pip` from within the virtualenv to install Tentacle:

```
(tentacle_env)$ pip install tentacle-0.1.0b.tar.gz
```

You can also install Tentacle from a downloaded clone of the **‘Bitbucket repository’_**. Assuming that the repository has been cloned to a folder `./tentacle` in your current directory, you can install it using `pip` like by issuing this command:

```
(tentacle_env)$ pip install ./tentacle
```

Using Tentacle without installation

It is also possible (but not recommended) to run tentacle without installing it into a virtualenv. To do this, unpack the archive and add the files in `$TENTACLE/rundir` to your `$PATH` variable. This could be done for your current user with the following commands:

```
$ tar -xvf tentacle-0.1.0.tar.gz  
$ ln -s tentacle-0.1.0/rundir/tentacle* ~/bin
```

This should work with a fresh clone of the **‘Bitbucket repository’_** as well. But please note that this is NOT the recommended way to use Tentacle.

Sequence alignment/mapping software

Adding support for other mappers

Tentacle is designed to make it simple to add support for additional mapping tools. The section *Creating/modifying mapper modules* contains instructions for how to extend the functionality of Tentacle with support for other CLI-based mappers.

To use Tentacle a sequence alignment software is required. In this documentation they will be referred to as ‘mapper’ or ‘sequence alignment software’ interchangeably. Tentacle comes with out-of-the-box support for the following mappers:

- [Bowtie2](#) (2.1.0)
- [GEM](#) (1.376 beta)
- [pBLAT](#) (v.34)
- [RazerS3](#) (3.2)
- [USEARCH](#) (v7.0.1001)
- [\(NCBI BLAST\)](#) (2.2.28+) *[not recommended: very slow]*

For installation instructions for the alignment software, please refer to the respective documentation/website.

After downloading/compiling the binaries for your mapper of interest, either ensure that they are available in `$PATH` or put the binaries (or symlinks) in `%TENTACLE_VENV%/bin` so that Tentacle can find them on runtime.

Tutorial

This part of the documentation shows examples of how to apply Tentacle to quantifying the contents of metagenomic samples. The sections below showcase how Tentacle can be used in different mapping scenarios, depending on the biological question.

The tutorials showcased here come with prepared example files that are available for download in section [Download](#). Note that the tutorial file contains example files for all tutorial examples showcased below, to provide a complete tutorial experience.

This tutorial is available online at: <http://bioinformatics.math.chalmers.se/tentacle/tutorial.html>

Important information about files and file formats

The Tentacle pipeline requires three types of data files:

- Reads (query sequences)
- Reference sequences
- Annotation for the reference sequences

Reads

Reads (or query sequences) are normally metagenomic reads produced with high-throughput sequencing technologies such as Illumina. The reads can be supplied to Tentacle as either FASTQ (with quality scores) or FASTA files. The

Tentacle pipeline can figure out what to do with either, but be aware that quality filtering cannot be performed and will be skipped for input files in FASTA format (it lacks quality information).

It is important to note that the input read files should preferably be located on a parallel file system that has high throughput and can handle the high load that will be put on it when Tentacle is running.

References

Reference sequences are normally some kind of gene or predicted ORF data. Tentacle expects reference sequences to be available in FASTA format, as this is the format used when indexing the reference sequences for coverage calculations.

Note: It is important that the reference filename ends in '.fasta', as Tentacle expects this filename ending and using reference files not ending in '.fasta' will result in an error.

Certain mappers require that the reference sequences are available in a custom database format (e.g. USEARCH uses *.udb and bowtie2 has several indexes in *.bt2.* files). It is important to note that you must prepare the reference sequences (FASTA file + potential database-related files) so that they share the same **basename** (i.e. the filename up until the first dot "."). For example, in the USEARCH case, create a tarball containing the following files:

```
$ ls
references.fasta references.udb
$ tar -zvcf reference_tarball.tar.gz references.fasta references.udb
```

It does not matter what you call the tarball, as long as the basename for the FASTA file and the database-related files are the same (in the above example the basename is *references*). The basename is important to remember to add to the command line arguments when running Tentacle so that the mapper can find the correct files later, e.g. for USEARCH the command line flag is `--usearchDBName`. Refer to the command line help for each mapper for their specific flag name.

Annotations

The annotations are used in Tentacle when producing and computing the coverage output. Tentacle computes the coverage (i.e. how large portions of the reference sequences that have reads aligned to them) and requires the annotation of the reference sequences to produce that output.

The annotation file is a simple tab separated text file with one annotated region per line. The format of the annotation file is as follows:

reference_name	start	end	strand	annotation
[ascii; no space]	[int]	[int]	[+ or -]	[ascii; no space]

The first few lines of an example annotation file could read:

scaffold3_2	899	3862	+	COG3321
scaffold6_1	0	570	+	COG0768
scaffold11_2	3	1589	-	NOG08628
scaffold13_1	1	260	-	NOG21937
scaffold13_1	880	1035	+	COG0110

As you can see in the example above, a reference sequence can occur on multiple lines, with different annotations on each line. Note that the start and end coordinates are 0-based (like Python).

Note: The annotation file is required in order to compute the coverage of each annotated region of the reference sequences. Tentacle will not run without an annotation file.

There is a special case where the entire length of each reference sequence is the actual annotated region (e.g. when the reference file contains entire genes). In such cases it is easy to create a dummy annotation file that annotates the entire length of each sequence in the reference FASTA file. Just put a + in the strand column.

TUTORIAL 1. Mapping reads to contigs (pBLAT)

This mapping scenario is relevant for quantifying the gene content of a complete metagenome. In this tutorial the mapper pBLAT will be used. However, the techniques displayed in this tutorial applies equally to other mappers that do not require a premade database (i.e. that can map a FASTA/FASTQ reads file to a FASTA reference), such as for example RazerS3.

First of all, make sure to install pBLAT and make the binary available in your \$PATH, e.g. by putting it in %TENTACLE_VENV/bin.

Step-by-step tutorial

To begin this tutorial, extract the tutorial tarball, available from [Download](#). It contains a folder named `tutorial_1` which contains the following files that are relevant for this part of the tutorial:

tutorial_1/	
tutorial_1/data/annotation_1.tab	tab-delimited file with annotation for contigs_
↪1.fasta	
tutorial_1/data/annotation_2.tab	tab-delimited file with annotation for contigs_
↪2.fasta	
tutorial_1/data/reads_1.fasta	reads in FASTA format
tutorial_1/data/reads_2.fastq	reads in FASTQ format
tutorial_1/data/contigs_1.fasta	contigs in FASTA format
tutorial_1/data/contigs_2.fasta	contigs in FASTA format

In our example, we are mapping reads from two small sequencing projects back to the contigs that were assembled from the same reads. One of the input read files is in FASTQ format, and one is in FASTA.

Step 1: Setting up the mapping manifest

For Tentacle to know what to do, a *mapping manifest* must be created. The manifest details what reads file should be mapped to what reference using what annotation. By utilizing a mapping manifest file, it is easy to go back to old runs and inspect their mapping manifests to see what was actually run.

The format for the mapping manifest is simple; it consists of three columns with absolute paths for the different files in the following order:

{reads}	{reference}	{annotation}
---------	-------------	--------------

To create a mapping manifest is easy. The simplest way is probably to use the standard GNU tools `find` and `paste`. Assuming you are standing in the `tutorial_1` directory it could look like this:

```
$ find `pwd`/data/r* > tmp_reads
$ find `pwd`/data/c* > tmp_references
$ find `pwd`/data/a* > tmp_annotations
```

```
$ paste tmp_reads tmp_references tmp_annotations > mapping_manifest.tab
$ rm tmp_*
```

What happens is that `find` lists all files matching the pattern `r*` in the data directory under our current working directory (`pwd` returns the absolute path to the current working directory), i.e. all read files in the data directory. We then do the same for the references (contigs in this case) and the annotation files. After we have produced three files containing listings of the absolute paths of all our data files, we paste them together using `paste` into a tab separated file `mapping_manifest.tab`.

This technique can easily be extended to add files from different folders by appending (`>>`) to the `tmp_reads` for example. There is no need to follow this specific procedure for the creation of the mapping manifest; you are free to use whatever tools or techniques you want for the mapping manifest as long as the end result is the same. It must contain absolute paths to all files and each row should contain three entries with read, reference, and annotation file.

Step 2: Run Tentacle on cluster using Slurm

Running Tentacle locally

Tentacle can also be run locally, with several instances of the mapper run simultaneously on your computer. This is not recommended as this is normally not very efficient, because several instances of the mapper will compete for resources (disk I/O, memory, CPU). To run Tentacle locally, call the file `tentacle_local.py` instead of `tentacle_slurm.py`.

As pBLAT is only able to read FASTA format files, the reads file in FASTQ format needs to be converted. Tentacle does this automatically when it detects that we are using a mapper that does not accept FASTQ input. The user does not have to do anything here.

For this tutorial we will use the default settings that pBLAT uses for mapping. For a list of options that can be modified for the specific mapper module used in Tentacle, run Tentacle with the `--pblat --help` command line options. For options not available via the mapper module in Tentacle, please refer to pBLAT's command line help.

First of all, make sure that the Python virtualenv that we created in the [Python virtualenv](#) section is activated. Tentacle can be run on the commandline by calling the file `tentacle_parallel.py`. If you installed Tentacle according to the instructions in [Download and install Tentacle](#) it should be available in your `$PATH` variable as well.

The call to Tentacle must minimally include the required command line parameters (in the case for pBLAT the only extra mapping related parameter required is the mapping manifest). If we use the mapping manifest that we created in Step 1, the command line could look like this:

```
$ tentacle_slurm.py --pblat --mappingManifest tutorial_1/mapping_manifest.tab --
↪distributionNodeCount 2 --slurmTimeLimit 01:00:00 --slurmAccount ACCOUNT2014-0-000 -
↪-slurmPartition glenn
```

A call like this runs Tentacle using the [Tentacle Slurm](#), e.g. in a cluster environment. Read more about running Tentacle in section [Running Tentacle](#). Note that you have to adjust the arguments for parameters `--slurmAccount`, `--slurmPartition`, to fit the account and partition names applicable in your specific cluster environment. If you want to try out Tentacle running locally, see the sidebar in this section.

Step 3: Check results

After a successful run, the Tentacle master process shuts down when all nodes have completed their computations. The results are continuously written to the output directory (which is either specified when starting the run using the `--outputDirectory` command line option or into the default output directory `tentacle_output`). The output

directory contains one folder with log files and one folder with the actual quantification results, as well as a file called `run_summary.txt` that shows an overview of all jobs.

The Tentacle output format is further detailed in section [Tentacle output](#).

TUTORIAL 2. Mapping nucleotide reads to amino acid database (USEARCH)

This mapping scenario is common typically when a reference database (ref DB) of known genes exists (e.g. known antibiotic resistance genes). Since all metagenomic samples needs to be compared to the same reference genes, a single ref DB is constructed beforehand. This steps displayed in this tutorial are relevant for other mappers using a premade ref DB such as Bowtie2, GEM, BLAST etc.

Introductory remarks

Modification of mapper call

How the actual commandline is constructed in Tentacle is defined in the mapping modules, in this case `usearch.py`; the interested reader should have a look there to see how it is constructed. It is available for inspection in [USEARCH](#).

In this example we will use USEARCH as the mapper because of its excellent performance in the nucleotide-to-amino-acid mapping scenario (translated search). As we are only interested in identifying the best matches we will utilize the `usearch_local` algorithm and search both strands of the reads. We are interested in genes with high sequence identity to the references and will only pick the best hit. If we boil it down to what we would run on a single machine, the commandline might look like this:

```
$ usearch -usearch_local reads.fasta -db references.udb -id 0.9 -strand both -query_
→cov 1.0
```

Step-by-step tutorial

To begin this tutorial, extract the tutorial tarball, available from [Download](#). It contains a folder called `tutorial_2` which contains the following files that are relevant for this part of the tutorial:

tutorial_2/	
tutorial_2/data/annotation.tab	tab-delimited file with annotation for
→references.fasta	
tutorial_2/data/reads_1.fasta	reads in FASTA format
tutorial_2/data/reads_2.fastq	reads in FASTQ format
tutorial_2/data/references.fasta	references in FASTA format

Step 1: Preparing the ref DB

Prior to running Tentacle, we need to prepare the reference sequences into the format that `usearch` uses for reference databases: `udb`. Running the following command in the `tutorial_2` directory will produce a `usearch` database that we can use:

```
$ usearch -makeudb_usearch data/references.fasta -output data/references.udb
```

There is one more thing that is required; Tentacle requires both the database file (for `usearch` to do its thing) but also the original FASTA file for the references, as this is used when computing the coverage of the reference sequences. So package all of the reference files (database and FASTA) into one `tar.gz` archive so that Tentacle can transfer both of them at once:

```
$ tar -cvzf data/references.tar.gz data/references*
```

Note how the basename of all files are the same (this is important!). When we are calling Tentacle later, we will have to specify the common basename using the `--usearchDBName` command line parameter (see section [Step 3: Run Tentacle](#)).

Step 2: Setting up the mapping manifest

For Tentacle to know what to do, a *mapping manifest* must be created. The manifest details what reads file should be mapped to what reference using what annotation. By utilizing a mapping manifest file, it is easy to go back to old runs and inspect their mapping manifests to see what was actually run.

The format for the mapping manifest is simple; it consists of three columns with absolute paths for the different files in the following order:

```
{reads}    {reference}    {annotation}
```

To create a mapping manifest is easy. The simplest way is probably to use the standard GNU tools `find` and `paste` like in the previous example above. However, in the case when a single reference database is to be used there is an extra step to ensure that there are as many lines of with the path to the reference database and the annotation file as there are read files to be mapped. Assuming you are standing in the `tutorial_2` directory it could look like this:

```
$ find `pwd`/data/reads* > tmp_reads
$ find `pwd`/data/references.tar.gz | awk '{for(i=0;i<2;i++)print}' > tmp_references
$ find `pwd`/data/annotation.tab | awk '{for(i=0;i<2;i++)print}' > tmp_annotations
$ paste tmp_reads tmp_references tmp_annotations > mapping_manifest.tab
$ rm tmp_*
```

What happens is that `find` lists all files matching the pattern `reads*` in the data directory under our current working directory (`pwd` returns the absolute path to the current working directory), i.e. all read files in the data directory. For references and annotations it is a bit different in this use case with a single reference database and accompanying single annotation file. In the example above we pipe the output from `find` via `awk` to multiply the line with the path to the reference tarball and the annotation file two times so that we can paste all the temporary files together and have one row for each read file. After we have produced three files containing listings of the absolute paths of all our data files, we paste them together using `paste` into a tab separated file `mapping_manifest.tab`.

This technique can easily be extend to add files from different folders by appending (`>>`) to the `tmp_reads` for example. There is no need to follow this specific procedure for the creation of the mapping manifest; you are free to use whatever tools or techniques you want for the mapping manifest as long as the end result is the same. It must contain absolute paths to all files and each row should contain three entries with read, reference, and annotation file.

Step 3: Run Tentacle

In this example we will map reads to a common reference database using the mapper `usearch`. Assuming we want to find the best alignment for each read to the reference using a 90% identity threshold the commandline for Tentacle/USEARCH could be the following. Assume you are standing in the `tutorial_2` directory:

```
$ tentacle_slurm.py --usearch --usearchDBName references.fasta --usearchID 0.9 --
↪mappingManifest mapping_manifest.tab --distributionNodeCount 2
```

The call to Tentacle when using `usearch` must minimally include the following command line arguments:

- `--mappingManifest`
- `--usearch`
- `--usearchDBName`

For more information about the available command line arguments, call Tentacle with the `--help` argument to display a list of all available options.

Step 4: Check results

After a successful run, the Tentacle master process shuts down after all nodes have completed computations. The results are continuously written to the output directory (which is either specified when starting the run using the `--outputDirectory` command line option or into the default output directory `tentacle_output`). The output directory contains one folder with log files and one folder with the actual quantification results.

The Tentacle output format is further detailed in section [Tentacle output](#).

Other mapping scenarios

Different mappers are best suited for different mapping tasks. With Tentacle it is possible to select the mapper that works best for your specific mapping scenario. The table below lists some scenarios and examples of what mappers might be best suited.

Scenario	Mapper(s)	Comments
Reads to annotated contigs	pBLAT, RazerS3	Many small “reference” files, potentially different for each reads file. (e.g. assembled contigs). No precomputed reference DB.
Reads to nt reference	USEARCH, GEM, Bowtie2	GEM works well with very large reference DBs
Reads to aa reference	USEARCH	BLASTX-like scenario, <i>translated search</i>

Running Tentacle

The tutorials ([Tutorial](#)) contain information and instructions on how to prepare data for a run with Tentacle.

Entry points

Tentacle has two main entry points:

- `tentacle_slurm.py`
- `tentacle_local.py`

These executables are Python scripts. They are installed into your `%TENTACLE_VENV%/bin` directory if Tentacle is installed using `pip`. If you did not perform a `pip` installation they are located in `$TENTACLE_ROOT/rundir`. You can add symlinks to them in e.g. your `~/bin` directory to make them universally available on your system if this is the case.

These executables will display on-screen help in the console if run without any arguments. The user must first choose one of the available mapping tools by supplying the relevant command line option (e.g. `--pblat`). When a valid

mapper option has been supplied it is possible to display further help on all the remaining options by supplying `--help` on the command line. The options vary a bit depending on what mapper is selected, so read them carefully. Since options are prone to change with updates to Tentacle all of them will not be covered here.

Tentacle Slurm

If Tentacle is launched using `tentacle_slurm.py` it will use the Slurm resource manager to launch worker nodes. This will only work on machines/clusters that have Slurm installed. To run Tentacle on Slurm, the following command-line arguments are required in addition to all mapping-related arguments:

- `--slurmAccount` - to specify your Slurm account name
- `--slurmPartition` - to specify your Slurm partition
- `--slurmTimeLimit` - time limit for nodes allocated via Tentacle

It is possible to run Tentacle with the `--localCoordinator` and `--distributionNoDedicatedCoordinatorNode` options. These tell Tentacle to launch the master process on one of the worker nodes side-by-side with computations. Usually it is best to leave these options as is, because it enables a master process to continue to run even if the time for allocated nodes should run out.

It is possible to launch additional workers that will attach to a currently running master process. See the next section for information about how to create additional workers.

Create additional workers

When a master process is started, a bash shell script is created in the current working directory (i.e. the directory from which the command was run), named `create_additional_workers_{JOBNAME}.sh`. This script will launch one additional worker node each time it is submitted to Slurm via `sbatch`. The script is generated when the master process is started and contains all the information required for a worker node to be launched and connect to the currently running master process and start requesting jobs. Note that it is possible to adjust the script manually to adjust parameters like requested allocation time in the Slurm scheduler if the time requested was too much or too little.

The script to create additional workers is no longer required after the master process closes and can thus be manually deleted when this has occurred.

Tentacle local

If Tentacle is launched using `tentacle_local.py` it will launch worker nodes locally on your computer. This can be used for trying things out or making smaller runs that does not require a big computer cluster. Note that running a large parallel job in local mode will put severe strain on the computer's I/O facilities and since most mappers are configured by default to utilize all available CPU cores the system might become unresponsive for a while.

Tentacle Query Jobs

There is a third script installed with the two mentioned above, `tentacle_query_jobs.py`. This script is a small program that makes it possible to query a running Tentacle master process to get the status of on-going jobs, look at (eventual) error messages from failed jobs etc. The program contains its own help message and descriptions on how to use it. To see instructions and help documentation, run the program with the help flag:

```
tentacle_query_jobs.py --help
```

A normal invocation to get a list of currently registered jobs and their status could look like this (substitute for the IP address of the currently running master process:

```
tentacle_query_jobs.py tcp://127.0.0.1
```

It can be run for single queries (default) but also has a useful interactive mode for more continuous querying of running jobs. The interactive mode is activated with `--interactive`.

Tentacle output

The output from Tentacle is written to the output directory, which can be specified with `--outputDirectory`. The default output directory is called `tentacle_output` and will, after a finished run, contain two folders and one file:

```
[tentacle_output]$ ls
logs results run_summary.txt
```

The folder `logs` contains all the log files produced during the run, ready for inspection if something went wrong. The `results` folder will contain one file with results for each reads file (mapping job) in the run. The last file, `run_summary.txt`, contains an overview of all the jobs in the run.

Tentacle output format

The format of the output file that Tentacle produces is a tab separated text file with five fields: annotation, number of mapped reads, median number of mapped reads, mean number of mapped reads, standard deviation of mapped reads. The format is as follows (with TAB as separating character):

```
contig_annotation:startpos:endpos:strand  count  median  mean    stdev
[contig: string (no tabs or spaces)      ]  [int]   [float]  [float]  [float]
[annotation: string (to tabs or spaces)]
[startpos: int                           ]
[endpos : int                             ]
[strand: + or -                           ]
```

This format is easy to parse and make further analyses on. Since each reference sequence header (e.g. `scaffold3_2`) is separated with a `_` before the annotation name (e.g. `COGxxxxxx`) it is easy to separate the annotation name from the reference sequence header and extract further information like start and stop coordinates of the annotation. Statistics like number of mapped reads (count), median, mean, and standard deviation, of mapped reads to each annotated region of the reference are useful when making further analyses of the data.

An example of Tentacle output could look like this (with TABs instad of spaces):

```
scaffold3_2_COG3321:898:3862:+  9   9.0 9.0 0.0
scaffold6_1_COG0112:0:570:+   1   1.0 1.0 0.0
scaffold11_2_NOG08628:2:1589:-  8   8.0 8.0 0.0
scaffold13_1_COG0028:2:260:-   0   0.0 0.0 0.0
```

In this simple example the first COG3321 annotated region on scaffold3_2 has had 9 reads that mapped to it, for a median, mean and standard deviation of 9.0, 9.0, and 0.0, respectively.

Parsers

Tentacle contains several parsers that parse input files to produce the data structures required to hold the results (coverage, counts) but also parsers for the output files from mappers (e.g. `gem`, `razers3`, or `blast` tabular formats).

If additional mappers are to be added to the program, suitable parsers might also be required. Have a look at how the supplied parsers are implemented and write something similar for the specific format you require. Make sure to add the parser to the `tentacle.parsers.__init__.py`.

Parsers module

Mapper output parsers

blast8 tabular format

GEM format

RazerS3 format

SAM format

Parser to create coverage data structure

Coverage

Coverage is computed for annotated regions in the reference database and implemented as follows. Each sequence in the reference database is represented by an array of integers which are incremented and decremented by 1 for each alignment start and end position respectively. Coverage over each reference sequence is then calculated as the cumulative sum over the corresponding array. The calculation of coverage has a time complexity of $O(n + N)$, where n is the total number of bases in the reference database and N is the number of mapped reads. This is substantially faster than the naïve approach which for each mapped read increments the coverage for each mapped base, yielding a time complexity of $O(n + N*M)$, where M is the maximum number of bases per read.

Modifying how coverage is computed

The Tentacle modules that compute coverage are located in `tentacle/coverage`. They use the mapping data in the `contigCoverage` data structure that is populated in `tentacle/parsers/index_references.py`. Check the code in that file to see how the dictionary is laid out. Essentially the dictionary holds a NumPy array for each of the sequences in the reference file. The array contains integers and after going through the mapper output each position in the array contains a number representing the number of times that position was covered by a read.

It is possible to modify the way the statistics are computed. See the files in the `coverage` module to see how it works.

Functions in the coverage module

Coverage

This module contains all the functions required to manipulate the `contig coverage` data structure.

Statistics

This module contains the function that computes statistics across annotated regions of the reference sequences.

Compute and write coverage statistics

This module contains a single function responsible for formatting the output.

Customizing modules in Tentacle

Tentacle is constructed using a modular approach making it easy to customize and modify. A common modification or addition to Tentacle could be to add support for an additional mapping software.

Creating/modifying mapper modules

Within the Tentacle folder structure, the mapper modules are located in `$TENTACLE_ROOT/tentacle/mappers`. If Tentacle was installed using `pip` into a virtualenv, the mapper modules must be placed (or symlinked) in `$TENTACLE_VENV/lib/python2.7/site-packages/tentacle/mappers` so that Tentacle can find them. Do not forget to add an import line in the `tentacle/mappers/__init__.py` as well!

Mapper modules inherit from the base mapper class, as seen in the example mapper modules that are provided with Tentacle. To create a new mapper module, copy one of the others, change the filename and modify it to suit your mapper. Make sure that the mapper you want to use is available in your `PATH` variable.

Another important aspect to consider when adding support for an additional mapper is that there needs to be a parser that can extract the information Tentacle requires from the mapper's output files. All such parsers are located in the submodule `tentacle.parsers`. See [Parsers](#).

Generic mapper class for Tentacle

Specific mapper classes for Tentacle

Here is a list of the implemented mapper classes that comes with Tentacle by default.

BLASTN

GEM

pBLAT

RazerS3

USEARCH

Choosing mapper

Bowtie2

Bowtie2 is a widely used mapping tool that is well suited for mapping reads to reasonably large databases and is commonly used for mapping to the human genome. As Bowtie2 requires a pre-constructed index its use in Tentacle is best suited when comparing to a single large database of reference sequences. It is implemented in Tentacle mainly as a reliable tool to perform filtering of human reads from metagenomic samples. Our testing showed that Bowtie2 was

unable to handle a case with a very large reference database (7.5 GiB in FASTA format). Bowtie2 is parallelized to take advantage of several CPUs.

GEM

GEM has implemented a mapping algorithm that also requires the construction of a pre-computed index. GEM was the only software in our testing that was capable of handling our largest test database weighing in at 7.5 GiB containing 11.5 million sequences in FASTA format. The GEM paper explains that GEM provides “fully tunable exhaustive searches that return all existing matches, including gapped ones”. GEM is parallelized and can take advantage of several CPUs.

BLAT

pBLAT is a parallelized version of the widely used BLAT tool. The parallelization allows pBLAT to utilize several CPUs on a single computer in a way that the original BLAT tool is unable to. The pBLAT website states “It can reduce run time linearly, and use almost equal memory as the original blat program”. As pBLAT does not require a precomputed index it is well suited for e.g. mapping metagenomic samples to their assembled contigs, which otherwise would require preparing many separate indexes if mapping with other mappers.

RazerS 3

RazerS 3 is a mapper that implements an algorithm that is well suited for aligning long reads with high error rates. RazerS 3 does not require a precomputed index, thus making it applicable in cases where metagenomic samples are mapped to their assembled contigs, just like e.g. pBLAT. The RazerS 3 publication shows that RazerS 3 often displays superior sensitivity when compared to other mappers. RazerS 3 is parallelised and can utilize several CPUs.

USEARCH

USEARCH is a widely used sequence alignment software that implements many BLAST-like sequence alignment options with a significantly higher speed and comparable sensitivity. The algorithm can create database indexes on-the-fly but can also precompute indexes to save time in cases where the same reference is used for all samples. USEARCH can utilize several CPUs and is capable of performing translated searches (e.g. searching with nucleotide reads against a protein database), making it very useful for quantifying the presence of specific genes in a metagenomic dataset.

BLAST

NCBI BLAST is a ubiquitous alignment algorithm that requires little introduction and capable of many different types of queries. It is not parallelized (however, as previously mentioned, some parallelized implementations such as mpiBLAST do exist). A mapping module for BLAST is included in Tentacle because it is widely used and the existence of such a module enables researchers that already use BLAST in their workflows to try out Tentacle without modifying too much of the workflow. However, we strongly discourage the use of BLAST because of its relatively slow speed compared to the previously described algorithms that can be hundreds of times faster than BLAST. Tentacle allows users that require the use of BLAST to essentially reduce the computation time by a factor of N, where N is the number of nodes used (as seen in the scaling evaluation).

Performance evaluation

The performance of Tentacle has been evaluated on its scaling characteristics when running in a distributed environment, as well as the quantification accuracy of the abundance estimation features. All evaluations and tests are described in the article (see [Citing Tentacle](#)).

Scaling performance

Tentacle scales very well with increasing computing resources. The following figure shows how the throughput of Tentacle scales when increasing the number of utilized worker nodes. For complete test details, refer to [Citing Tentacle](#). The evaluation results data and code to generate the figures are available for download from [figshare](#). A non-interactive version of the IPython notebook can be [viewed in your browser](#).

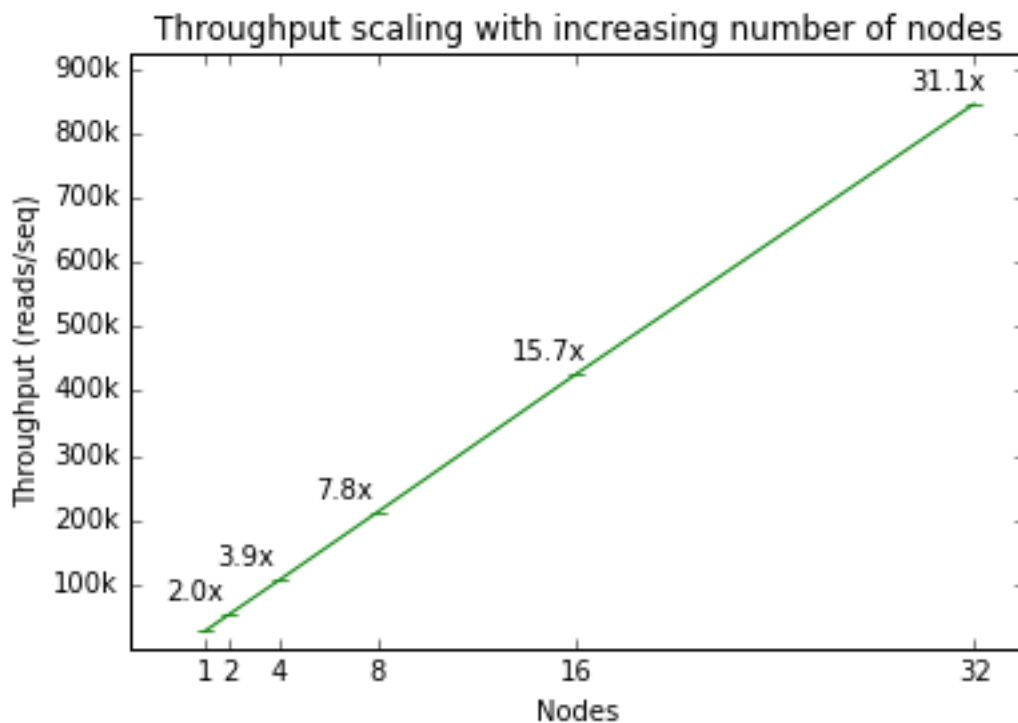


Fig. 2.3: Tentacle scales very well with increasing number of computer nodes.

Quantification accuracy

The accuracy of quantification and coverage was also estimated. The following two figures show the expected and measured coverage and quantification results. For complete test details, please refer to [Citing Tentacle](#) and the IPython notebook which contains the code to generate the figures. The evaluation results data and code to generate the figures are available for download from [figshare](#). A non-interactive version of the IPython notebook can be [viewed in your browser](#).

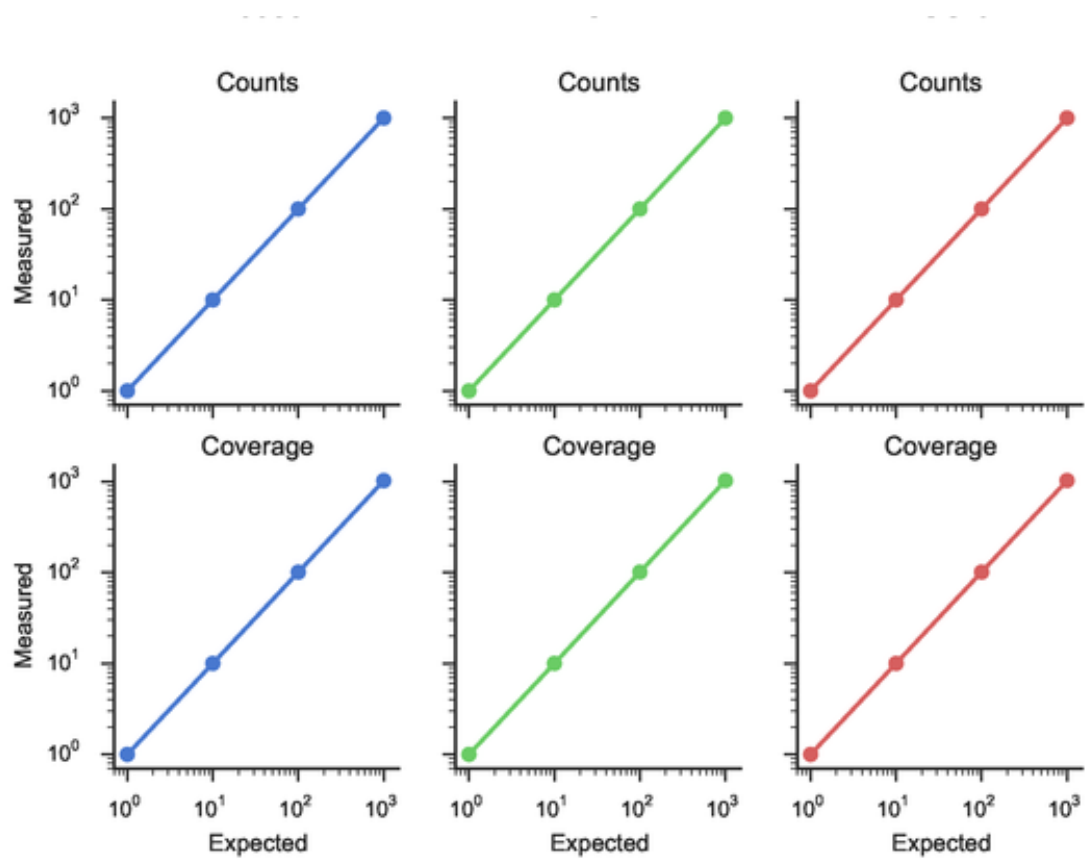


Fig. 2.4: Tentacle achieves very good coverage and quantification accuracy.

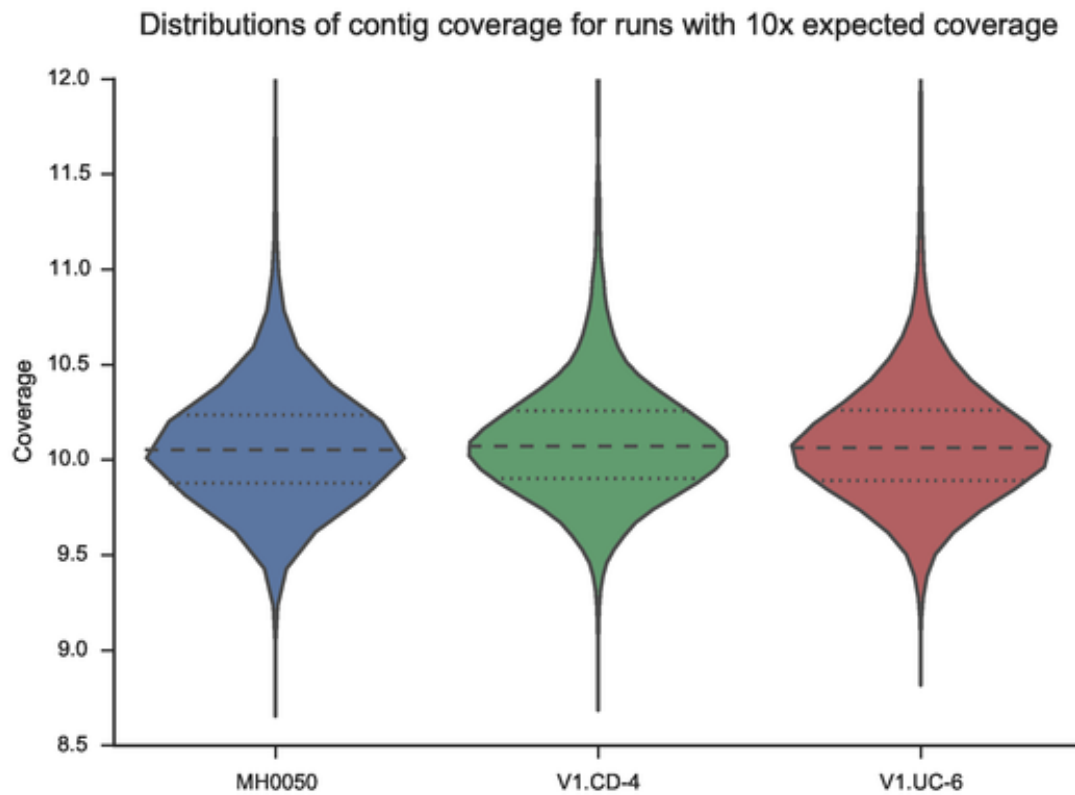


Fig. 2.5: The distribution of the measured quantification levels follows the expected values for this synthetic test case.

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)