itero Documentation

Release 1.1

Brant C. Faircloth

Jun 08, 2018

Contents:

Release v1.1. (*Changelog*)

Author Brant C. Faircloth

Date 08 June 2018 15:20 UTC (+0000)

itero is a "guided-assembly" workflow for target enrichment data that uses bwa, samtools, bedtools, and spades to provided high-quality assemblies of raw reads from Illumina instruments.

CHAPTER 1

Contributions

itero is open-source (see *License*) and we welcome contributions from anyone who is interested. Please make a pull request on github. The issue tracker for itero is also available on github.

CHAPTER 2

Issues

If you have an issue, please ensure that you are experiencing this issue on a supported OS (see *Installation*) using the conda installation of itero. If possible, please submit a test case demonstrating the issue and indicate which platform, git checkout, and phyluce version you are using.

CHAPTER 3

Guide

3.1 Purpose

itero is a program what wraps a workflow for guided- or reference-based assembly of target enrichment data. This approach to assembly is also "iterative", meaning that the assembly proceeds through several, iterations (hence "itero"). I wrote itero for a variety of reasons:

- "traditional" DNA assembly programs performed poorly with target enrichment data (from UCE loci)
- existing DNA assembly approaches had relatively high assembly error
- existing guided assembly programs were hard to install and run
- some existing guided assembly programs were **slow**

itero attempts to fix some of these problems. At its heart, itero uses an input file of "seeds", against which it will try to assemble raw-read data from Illumina instruments. Alignment of reads-to-seeds uses bwa, the BAM file is split with samtools and bedtools, and locus-specific reads are then assembled using spades (with error correction turned on during the final round). Then, the entire process repeats itself.

To increase assembly speed, itero takes advantage of multiple cores (on single nodes) using python multiprocessing and MPI (on HPC systems) using the excellent schwimmbad library.

3.1.1 Who wrote this?

This documentation was written primarily by Brant Faircloth (http://faircloth-lab.org). Brant is also responsible for the development of most of the itero code. Bugs within the code are usually his.

3.1.2 How do I report bugs?

To report a bug, please post an issue to https://github.com/faircloth-lab/itero/issues. Please also ensure that you are using one of the "supported" platforms:

- Apple OSX 10.9.x

- CentOS 7.x

and that you have installed itero and dependencies using conda, as described in the *Installation* section.

3.2 Installation

itero uses a number of Python tools that allow it to assemble raw reads into contigs. itero also wraps a number of third-party programs. These include:

3.2.1 Python Modules

- numpy
- biopython
- mpi4py
- schwimmbad
- six

3.2.2 3rd-party programs

- bedtools
- bwa
- gawk
- samtools
- spades

To ensure that these dependencies are easy to install, we have created a conda package for itero that is available as part of bioconda. This is the easiest way to get itero up and running on your system. itero can also be run outside of conda, and we include some installation suggestions for these types of systems, below. However, because many HPC systems are configured differently, we cannot provide extensive support for itero on HPC platforms.

Note: We build and test the binaries available through conda using 64-bit operating systems that include:

- Apple OSX 10.9.x
- CentOS 7.x

3.2.3 Why conda?

It may seem odd to impose a particular distribution on users, and we largely agree. However, conda makes it very easy for us to distribute both Python and non-Python packages, setup identical environments across very heterogenous platforms (linux, osx), make sure all the *$PATHs* are correct, and have things run largely as expected. Using conda has several other benefits, including environment separation similar to virtualenv.

In short, using conda gets us as close to a "one-click" install that we will probably ever get.

3.2.4 Install Process (using conda/bioconda)

Attention: We do not support itero on Windows.

Note: We build and test the binaries available through using 64-bit operating systems that include:

- Apple OSX 10.9.x
- CentOS 7.x

The installation process is a 3-step process. You need to:

1. Install conda (either anaconda or miniconda)
2. Configure conda to use bioconda
3. Install itero

Installing itero using conda will install all of the required binaries, libraries, and Python dependencies.

Install Anaconda or miniconda

You first need to install anaconda or miniconda. Which one you choose is up to you, your needs, how much disk space you have, and if you are on a fast/slow connection.

Attention: You can easily install anaconda or miniconda in your $HOME, although you should be aware that this setup can cause problems in some HPC setups.

Tip: Do I want anaconda or miniconda?

The major difference between the two python distributions is that anaconda comes with many, many packages pre-installed, while miniconda comes with almost zero packages pre-installed. As such, the beginning anaconda distribution is roughly 500 MB in size while the beginning miniconda distribution is 15-30 MB in size.

anaconda

Follow the instructions here for your platform: http://docs.continuum.io/anaconda/install.html

miniconda

Find the correct *miniconda-x.x.x* file for your platform from http://repo.continuum.io/miniconda/ and download that file. Be sure you **do not** get one of the packages that has a name starting with *miniconda3-*. When that has completed, run one of the following:

```
bash Miniconda-x.x.x-Linux-x86_64.sh  [linux]
bash Miniconda-x.x.x-MacOSX-x86_64.sh [osx]
```

Note: Once you have installed Miniconda, we will refer to it as **anaconda** throughout the remainder of this documentation.

Checking your *$PATH*

Regardless of whether you install anaconda or miniconda, you need to check that you've installed the package correctly. To ensure that the correct location for anaconda or miniconda are added to your $PATH (this occurs automatically on the $BASH shell), run the following:

```
$ python -V
```

The output should look similar to (*x* will be replaced by a version):

```
Python 2.7.x :: Anaconda x.x.x (x86_64)
```

Notice that the output shows we're using the *Anaconda x.x.x* version of Python. If you do not see the expected output (or something similar), then you likely need to edit your $PATH variable to add anaconda or miniconda.

The easiest way to edit your path, if needed is to open `~/.bashrc` with a text editor (if you are using ZSH, this will be `~/.zshrc`) and add, as the last line:

```
export PATH=$HOME/path/to/conda/bin:$PATH
```

where `$HOME/path/to/conda/bin` is the location of anaconda/miniconda on your system (usually `$HOME/anaconda/bin` or `$HOME/miniconda/bin`).

Warning: If you have previously set your `$PYTHONPATH` elsewhere in your configuration, it may cause problems with your anaconda or miniconda installation of phyluce. The solution is to remove the offending library (-ies) from your `$PYTHONPATH`.

Configure Bioconda

Once you have installed anaconda (or miniconda), you need to configure conda to use the bioconda channel. More information on this process can be found on the bioconda website, but the gist of the process is that you need to run:

```
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
```

Install itero

Once bioconda is installed, you should be able to install itero by running:

```
conda install itero
```

This will install everything that you need to run the program.

Test itero install

You can check to make sure all of the binaries are installed correctly by running:

```
itero check binaries
```

3.2.5 Install Process (Alternative / HPC)

On some systems (particularly HPC systems), conda can cause problems. You can itero the "old" way by downloading the package tarball (https://github.com/faircloth-lab/itero/releases) and running:

```
python setup.py install
```

in the main directory. This should install all of the Python dependencies, **but you still need to install and configure the 3rd-party dependencies**.

> **Attention:** You will need to install 3rd-party dependencies on your own if you are using the *python setup.py install* method of installing itero

You can build and install these dependencies where you like. To configure itero to use the dependencies you have build and installed, you need to create a `$HOME/.itero.conf` that gives the paths to each program and looks like:

```
[executables]
bedtools:/path/to/bin/bedtools
bwa:/path/to/bin/bwa
gawk:/path/to/bin/gawk
samtools:/path/to/bin/samtools
spades:/path/to/bin/spades.py
```

Test itero install

You can check to make sure all of the binaries are installed correctly by running:

```
itero check binaries
```

3.3 Running itero

itero has both a **local** mode and an **MPI** mode. The **local** mode is for execution on a single node, while the **MPI** mode executes individual locus assemblies in parallel using an MPI-enabled HPC cluster. To run the program, you must first create a configuration file denoting the samples you wish you assemble. That file has the following format:

```
[reference]
/path/to/the/locus/seeds.fasta

[individuals]
taxon-one:/path/to/fastq/R1/and/R2/files/for/taxon/1/
taxon-two:/path/to/fastq/R1/and/R2/files/for/taxon/2/
taxon-three:/path/to/fastq/R1/and/R2/files/for/taxon/3/
```

3.3.1 itero on a single node

You then run the *local* version using a command similar to:

```
itero assemble local --config ndna-test.conf
    --output local
    --local-cores 16
    --iterations 6
```

This will run *itero* on a single node and will first use 16 cores to perform *bwa* alignments. The code will then distribute locus-specific assemblies across all cores on the node (1 assembly per core; 16 in parallel).

3.3.2 itero across MPI nodes

You run the *MPI* version using a command similar to:

```
mpirun -hostfile hostfile -n 96 itero assemble mpi --config ndna-test.conf \
    --output mpi \
    --local-cores 16 \
    --iterations 6
```

If each of your nodes has 16 cores, this will first use 16 cores for the needed *bwa* alignments of reads to seeds. The code will then distribute locus-specific assemblies across all 96 cores in your cluster (1 assembly per core; 96 in parallel).

CHAPTER 4

Project info

4.1 License

4.1.1 Documentation

The documentation for itero is available under a CC-BY (2.0) license. This license gives you permission to copy, distribute, and trasmit the work as well as to adapt the work or use this work for commercial purposes, under the condition that you must attribute the work to the author(s).

If you use this documentation or the itero software for your own research, please cite both the software and (Faircloth et al. 2012). See the Citing section for more detail.

4.1.2 Software

TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.2 Changelog

4.2.1 v1.0.x (April 2018)

- initial version with MPI and multiprocessing capability

4.2.2 v1.1.0 (May 2018)

- fix error in contig checking code that could cause MPI operations to hang
- refactor BAM splitting code for hopefully faster operation
- add RAM limits on spades
- add configuration parameters to iter.conf for spades
- create unique log file for each run

4.2.3 v1.1.1 (June 2018)

- fix an error where too many fastq files would cause MPI to hang

4.3 Funding

4.3.1 Primary Sources

The National Science Foundation (NSF) has supported a large portion of our work. The specific programs and proposal identifiers are below:

- NSF DEB-1655624
- NSF IOS-1754417