MetAMOS Documentation

Release 1.5rc1

MetAMOS development team

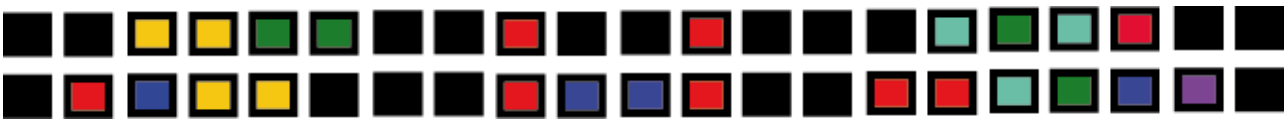
Aug 21, 2017

Contents

1 Overview **3**

2 Citation **5**

2.1 Hardware requirements . . . 5
2.2 Installation . . . 6
2.3 Single binary . . . 9
2.4 Test suite . . . 10
2.5 Quick Start . . . 12
2.6 iMetAMOS . . . 17
2.7 Workflows . . . 17
2.8 Generic tools (or plug-in framework) . . . 19
2.9 MetAMOS directory structure . . . 22
2.10 Output . . . 31
2.11 Supported Programs . . . 34
2.12 FAQs . . . 37
2.13 Contact . . . 39
2.14 Experimental: TweetAssembler v0.1b . . . 39



CHAPTER 1

Overview

MetAMOS represents a focused effort to create automated, reproducible, traceable assembly & analysis infused with current best practices and state-of-the-art methods. MetAMOS for input can start with next-generation sequencing reads or assemblies, and as output, produces: assembly reports, genomic scaffolds, open-reading frames, variant motifs, taxonomic or functional annotations, Krona charts and HTML report.

CHAPTER 2

Citation

If you use MetAMOS in your research, please cite the following manuscript (in addition to the individual software component citations listed in stdout):

```
TJ Treangen, S Koren, DD Sommer, B Liu, I Astrovskaya, B Ondov, Irina Astrovskaya,␣
↪Brian Ondov, Aaron E Darling, Adam M Phillippy, Mihai Pop
MetAMOS: a modular and open source metagenomic assembly and analysis pipeline
Genome Biology 14 (1), R2
```

Contents:

Hardware requirements

MetAMOS was designed to work on any standard 64bit Linux or OSX environment. To use MetAMOS for tutorial/teaching purposes, a minimum of 16 GB RAM is recommended. To get started on real data sets a minimum of 64 GB of RAM is recommended, and up to 1 TB of RAM may be necessary for larger datasets. In our experience, for most 50-100 million read datasets, 64-128 GB is a good place to start.

Scenario #1: running locally on large memory server

Suggested all-purpose build:

- 256 GB RAM (16 x 16GB)
- 48 cores (96 HT, 4x cpu)
- 1 TB SSD temporary scratch space for running analyses
- 16 TB HDD archival space for storing analyses

Scenario #2: running on local cluster/grid

Notes:

- Great for BLAST intense jobs
- RAM will limit the supported assemblers
- Grid job submission via SGE, others not supported
- MPI install required for Ray Meta

Scenario #3: running on cloud via Amazon EC2 High Memory

Recommend for best price/performance ratio -> cr1.8xlarge (Memory optimized):

- 2 X 120 GB SSD
- 32 HT x 2.8GhZ
- 244 GB RAM
- Spot instance currently at $0.361 per Hour (based on availability)
- On demand instance at $3.500 per Hour (always available)
- Reserved instance at $1.54 per Hour, $2474 upfront, approx. $2000 a month

More info on spot instances: http://aws.amazon.com/ec2/purchasing-options/spot-instances/

Or for smaller assembly jobs -> c3.8xlarge (Compute optimized):

- 2 X 320 GB SSD
- 32 HT x 2.8GhZ
- 60 GB RAM
- On demand instance at $2.400 per Hour (spot instance same price!)

Here is a very useful cost calculator: https://www.scalyr.com/cloud/

Don't forget to account for time/cost to upload data!

Installation

Before your start

The most common cause of a failed run is a missing package or dependency. We provide two paths to simplify the task of downloading and manual install of all required dependencies: an install script (INSTALL.py) and a frozen binary. See below sections for further details.

Prerequisites

MetAMOS has several dependencies/prerequisites, the large majority of which are automatically downloaded and installed when running INSTALL.py (see next section). In addition, several dependences/prerequisites are not installed by INSTALL.py and must be available on your system:

- Java (6+)

- perl (5.8.8+)
- python (2.7.3+)
- R (2.11.1+ with PNG support)
- gcc (4.7+ for full functionality)
- curl
- wget

Here is a list of currently supported Operating Systems:

1. Mac OSX (10.7 or newer)
2. Linux 64-bit (tested on CentOS, Fedora, RedHat, OpenSUSE and Ubuntu)

And here is a current list of required packages/libs installed by metAMOS:

1. **Perl**
    - File::Copy::Link
    - Statistics::Descriptive
    - Time::Piece
    - XML::Parser
    - XML::Simple
2. **Python**
    - Cython
    - matplolib (1.3.0, newer versions may not work)
    - NumPy
    - psutil
    - PySam
    - setuptools
3. **Other**
    - Boost
    - cmake
    - Jellyfish
    - SparseHash

Automated installation

MetAMOS contains an automated installation script which installs MetAMOS along with required Python dependencies, third party software and necessary data files. If you encounter issues during installation, you can try installing the required dependencies manually and re-running INSTALL.py. If you continue to encounter issues, plase provide the output from INSTALL.py as a new issue.

To download the software release package:

```
$ wget https://github.com/marbl/metAMOS/archive/v1.5rc3.zip
```

If you see a certificate not trusted error, you can add the following option to wget:

```
$ --no-check-certificate
```

And if `wget` not available, you can use `curl` instead:

```
$ curl -L https://github.com/marbl/metAMOS/archive/v1.5rc3.zip > v1.5rc3.zip
```

You can also browse the https://github.com/marbl/MetAMOS/tree/v1.5rc3 and click on Downloads.

Once downloaded, extract to unpack:

```
$ unzip v1.5rc3.zip
```

Change to MetAMOS directory:

```
$ cd metAMOS-v1.5rc3
```

Once inside the MetAMOS directory, run:

```
$ python INSTALL.py core
```

This will download and install the external dependencies which may take minutes or hours to download depending on your connection speed. metAMOS supports workflows to install subsets of tools for faster installation. By default only the core dependencies are installed.

To install iMetAMOS run:

```
$ python INSTALL.py iMetAMOS
```

Also, you can run:

```
$ python INSTALL.py -h
```

to get a listing of available workflows and programs. You can specify either workflows or programs as arguments to INSTALL.py. For example, to install the core workflow plus PhyloSift, run:

```
$ python INSTALL.py core phylosift
```

To install the programs which are part of the optional workflow run:

```
$ python INSTALL.py optional
```

If all dependencies are downloaded (including optional/deprecated ones), this will take quite awhile to complete (plan on a few hours to 2 days).

Running the test suite

MetAMOS comes with a comprehensive test suite to make sure that installation has succeeded on your system. To run a quick test and very installation succeeded run:

```
$ cd ./Test
$ ./run_pipeline_test.sh
```

Single binary

MetAMOS PyInstaller single file binary

In attempt to further simplify the MetAMOS installation process, we are happy to announce the availability of a 'frozen' MetAMOS binary for Linux-x68_64 platforms. Along with this binary comes a significantly reduced list of prerequisites:

- Java 1.6 (or newer)
- **Perl 5.8.8**
  - **Newer versions of Perl are not backwards compatible so if you have Perl 5.10 (or newer) you may need to install**
    - Statistics::Descriptive
    - Bio::Seq
    - Time::Piece
    - XML::Simple
    - Storable
    - XML::Parser
    - File::Spec
- R 2.11.1 (or newer)
- 64-bit *nix OS or Mac OSX 10.7+ (you may need to install MacPorts for full functionality)

Disclaimer: The frozen binary is provided as-is and has limited support and reduced functionality/features compared to installing from source. If you encouter issues with a frozen binary, please try installing the latest release.

First, select your flavor (DBs below are required but provided separately):

Linux 64-bit:

```
$ wget ftp://ftp.cbcb.umd.edu/pub/data/treangen/MA_fb_v1.5rc2_linux.tar.gz
$ tar -xf MA_fb_v1.5rc2_linux.tar.gz
$ chmod u+rwx metAMOS_v1.5rc2_binary
```

OSX 64-bit:

```
$ wget ftp://ftp.cbcb.umd.edu/pub/data/treangen/MA_fb_v1.5rc2_OSX.tar.gz
$ tar -xf MA_fb_v1.5rc2_OSX.tar.gz
$ chmod u+rwx metAMOS/*
```

Then add the toppings. The light DB is recommended if you are testing/getting started with metAMOS installation. If you are planning to do analysis, the full DB download is recommended. The full DB adds support for:

- FCP classifier
- BLAST databases required for BLAST-based classification
- RefSeq database for required to recruit references for validation

When using the miniature DB, some features will be automatically disabled. Only Kraken can be used as the classifier with its miniature database and QUAST cannot be used as no reference will be available for recruitment. This should

be a download once and only once operation. Updated frozen binaries will be backwards compatible with a previous DB download. Further details on the expected DBs on the readthedocs page.

ALL DBS:

```
$ wget ftp://ftp.cbcb.umd.edu/pub/data/treangen/allDBs.tar.gz
$ tar -xf ftp://ftp.cbcb.umd.edu/pub/data/treangen/allDBs.tar.gz -C [$METAMOS_BIN_
↪INSTALL_DIR]/
```

LIGHT DBS:

```
$ wget ftp://ftp.cbcb.umd.edu/pub/data/treangen/minDBs.tar.gz
$ tar -xf ftp://ftp.cbcb.umd.edu/pub/data/treangen/minDBs.tar.gz -C [$METAMOS_BIN_
↪INSTALL_DIR]/
```

Finally, run a quick test:

```
$ cd ./Test
$ ./run_pipeline_test.sh
```

The frozen binary is actually a collection of programs that extracts/runs/cleans up automatically using PyInstaller. By default, PyInstaller will use the following directories to extract into:

- The directory named by the TMPDIR environment variable.
- The directory named by the TEMP environment variable.
- The directory named by the TMP environment variable.

If your system is missing all of the above, does not have sufficient space, or is missing write-premissions, runPipeline will not be able to extract itself and will report: INTERNAL ERROR: cannot create temporary directory!. The extracted runPipeline requires at least 4GB of free temporary disk space. You will get a "No DBs found ERROR!" if you do not download any DBs. The DB dir needs to be placed inside of the frozen binary install dir.

Note: please use caution! this binaries eat up disk space quickly. Please ensure you have ample free space (100GB+) before download & use.

Test suite

Test scripts and sanity checks

We have developed a set of scripts for testing the various features of MetAMOS. All of these regression test scripts are available inside the /Test directory and include all necessary datasets to run them. Here is a brief listing of the test scripts we currently include:

Test initPipeline

./Test/test_create.sh

Test runPipeline

./Test/run_test.sh

Test Preprocess filtration of non-interleaved fastq files

./Test/test_filter_noninterleaved_fastq.sh

Test iMetAMOS

./Test/test_ima.sh

Test SRA download

./Test/test_sra.sh

Test Newbler (if available)

./Test/test_newbler.sh

Test CA (fasta)

./Test/test_ca_fasta.sh

Test CA (fastq)

./Test/test_ca.sh

Test SOAPdenovo

./Test/test_soap.sh

Test MetaVelvet

./Test/test_metavelvet.sh

Test SparseAssembler

./Test/test_sparse.sh

Test Velvet

./Test/test_velvet.sh

Test FCP

./Test/test_fcp.sh

Test Spades

./Test/test_spades.sh

Test BLAST

./Test/test_blast.sh

Quick Start

Getting started

Before you get started using MetAMOS/iMetAMOS a brief review of its design will help clarify its intended use. MetAMOS gas two main components:

1. initPipeline
2. runPipeline

initPipeline

The first component, initPipeline, is for creating new projects and also initializing sequence libraries. Currently interleaved & non-interleaved fasta, fastq, and SFF files are supported. Input files can be compressed (bzip2, gzip) and can reside on remote servers (in this case the full URL must be specified). SRA run identifiers are also supported.

The file-type flags (-f, -q, and -s) must be specified before the file. Once specified, they remain in effect until a different file type is specified.

usage: initPipeline -f/-q -1 file.fastq.1 -2 file.fastq.2 -d projectDir -i 300:500

options

The following options are supported:

```
-1: either non-paired file of reads or first file in pair, can be list of multiple␣
↪separated by a comma
-2: second paired read file, can be list of multiple separated by a comma
-c:  fasta file containing contigs
-d: output project directory (required)
-f: boolean, reads are in fasta format (default is fastq)
-h: display help message
-i: insert size of library, can be list separated by commas for multiple libraries
-l: SFF linker type
-m: interleaved file of paired reads
-o: reads are in outtie orientation (default innie)
-q: boolean, reads are in fastq format (default is fastq)
-s/--sff: boolean, reads are in SFF format (default is fastq)
-W: string: workflow name (-W iMetAMOS will run iMetAMOS).
```

A workflow can specify parameters as well as data. A workflow can be immutable in which case any command-line parameters will not be used. Otherwise, command-line parameters take priority over workflow defaults.

Common use-cases

- non-interleaved fastq, single library:

```
initPipeline -q -1 file.fastq.1 -2 file.fastq.2 -d projectDir -i 300:500
```

- non-interleaved fasta, single library:

```
initPipeline -f -1 file.fastq.1 -2 file.fastq.2 -d projectDir -i 300:500
```

- interleaved fastq, single library:

```
initPipeline -q -m file.fastq.12  -d projectDir -i 300:500
```

- interleaved fastq, multiple libraries:

```
initPipeline -q -m file.fastq.12,file2.fastq.12  -d projectDir -i 300:500,
↪1000:2000
```

- interleaved fastq, multiple libraries, existing assembly:

```
initPipeline -q -m file.fastq.12,file2.fastq.12 -c file.contig.fa -d projectDir -
↪i 300:500,1000:2000
```

- non-interleaved remote fastq, single library:

```
initPipeline -q -1 ftp://ftp.cbcb.umd.edu/pub/data/metamos/gage-b-rb.miseq.1.
↪fastq.gz -2 ftp://ftp.cbcb.umd.edu/pub/data/metamos/gage-b-rb.miseq.2.fastq.gz -
↪d projectDir -i 300:500
```

- unpaired SRA run using iMetAMOS:

```
initPipeline 1 <SRA RUN ID> -d projectDir -W iMetAMOS
```

- paired-end SRA run using iMetAMOS:

```
initPipeline -m <SRA RUN ID> -d projectDir -i 300:500 -W imetAMOS
```

runPipeline

The second component, runPipeline, takes a project directory as input and runs the following steps by default:

1. Preprocess
2. Assemble
3. FindORFs
4. Validate
5. FindRepeats
6. Abundance
7. Annotate
8. FunctionalAnnotation
9. Scaffold
10. Propagate
11. FindScaffoldORFs

12. Classify
13. Postprocess

options

usage: runPipeline [options] -d projectdir:

```
-h = <bool>:   print help [this message]
-j = <bool>:   just output all of the programs and citations then exit (default = NO)
-v = <bool>:   verbose output? (default = NO)
-d = <string>: directory created by initPipeline (REQUIRED)
```

[options]: [pipeline_opts] [misc_opts]

[pipeline_opts]: options that affect the pipeline execution

Pipeline consists of the following steps:

```
Preprocess (required)
Assemble
FindORFS
MapReads
Validate
Abundance
Annotate
Scaffold
Propagate
Classify
Postprocess (required)
```

Each of these steps can be referred to by the following options:

```
-f = <string>: force this step to be run (default = NONE)
-s = <string>: start at this step in the pipeline (default = Preprocess)
-e = <string>: end at this step in the pipeline (default = Postprocess)
-n = <string>: step to skip in pipeline (default=NONE)
```

For each step you can fine-tune the execution as follows

Preprocess

Preproces options:

```
-t = <string>:   enable filter of input reads (default = metAMOS, options = metAMOS,␣
↪EA-UTILS, PBcR for PacBio sequences)
-q = <bool>:     produce FastQC quality report for reads with quality information␣
↪(fastq or sff)? (default = NO)
```

Assemble

Assemble options:

```
-a = <string>:    Genome assembler to use (default = SOAPdenovo).
                  This can also be a comma-separated list of assembler (for example:␣
↪soap,velvet)
                  in this case, all selected assemblers will be run and the best␣
↪selected for subsequent analysis

-k = <kmer size>: k-mer size to be used for assembly (default = auto-selected).
                  This can also be a comma-separated list of kmers to use

-o = <int>:       min overlap length
```

MapReads

MapReads options:

```
-m = <string>:    read mapper to use? (default = bowtie)
-i = <bool>:      save bowtie (i)ndex (default = NO)
-b = <bool>:      create library specific per bp coverage of assembled contigs␣
↪(default = NO)
```

FindORFS

FindORFS options:

```
-g = <string>:    gene caller to use (default=FragGeneScan)
-l = <int>:       min contig length to use for ORF call (default = 300)
-x = <int>:       min contig coverage to use for ORF call (default = 3X)
```

Validate

Validate options:

```
-X = <string>:   comma-separated list of validators to run on the assembly. (default␣
↪= lap, supported = reapr,orf,lap,ale,quast,frcbam,freebayes,cgal,n50)
-S = <string>:   comma-separated list of scores to use to select the winning assembly.
↪ By default, all validation tools specified by -X will be run.
                 For each score, an optional weight can be specified as SCORE:WEIGHT.
                 For example, LAP:1,CGAL:2 (supported = all,lap,ale,cgal,snp,frcbam,
↪orf,reapr,n50)
```

Annotate

Annotate options:

```
-c = <string>:  classifier to use for annotation (default = FCP)
-u = <bool>:    annotate unassembled reads (default = NO)
```

Classify

Classify options:

```
-z = <string>: taxonomic level to categorize at (default = class)
```

[misc_opts]

Miscellaneous options:

```
-r = <bool>:   retain the AMOS bank  (default = NO)
-p = <int>:    number of threads to use (be greedy!) (default=1)
-4 = <bool>:   454 data (default = NO)
```

Common use-cases

- To enable read filtering:

```
-t
```

- To enable IDBA_ud as the assembler:

```
-a idba_ud
```

- To use Kraken for read classifcation:

```
-c kraken
```

- Any single step in the pipeline can be skipped by passing the following parameter to runPipeline:

```
-n,--skipsteps=Step1,..
```

- MetAMOS reruns steps based on timestamp information, so if the input

files for a step in the pipeline hasn't changed since the last run, it will be skipped automatically. However, you can forcefully run any step in the pipeline by passing the following parameter to runPipeline:

```
-f,--force=Step1,..
```

MetAMOS stores a summary of the input libraries in pipeline.ini in the working directory. The pipeline.conf file stores the list of programs available to MetAMOS. Finally, pipeline.run stores the selected parameters and programs for the current run. MetAMOS also stores detailed logs of all commands executed by the pipeline in Logs/COMMANDS.log and a log for each step of the pipeline in Logs/<STEP NAME>.log

Upon completion, all of the final results will be stored in the Postprocess/out directory. A component, create_summary.py, takes this directory as input and as output, generates an HTML page with with summary statistics and a few plots. An optional component, create_plots.py, takes one or multiple Postprocess/out directories as input and generates comparative plots.

iMetAMOS

What is iMetAMOS

iMetAMOS is an extension of metAMOS to isolate genome assembly. It is a workflow which, by default, uses multiple assemblers and validation tools to select the best assembly for a given sample. Effectively, this is equivalent to GAGE-in-a-box or ensemble assembly. iMetAMOS is included in the frozen binary.

If you have used iMetAMOS for analyzing your, please cite (in addition to the individual software component citations listed in main output):

```
Koren S, Treangen TJ, Hill CM, Pop M, Phillippy AM
Automated ensemble assembly and validation of microbial genomes.
BMC Bioinformatics 15:126, 2014.
```

Please also consider citing the original metAMOS publication:

```
Treangen TJ\*, Koren S\*, Sommer DD, Liu B, Astrovskaya I, Ondov B, Darling AE,␣
↪Phillippy AM, Pop M.
MetAMOS: a modular and open source metagenomic assembly and analysis pipeline.
Genome Biol. 2013 Jan 15;14(1):R2. PMID: 23320958.
```

*Indicates both authors contributed equally to this work

To install iMetAMOS without using a frozen binary, run:

```
$ curl -L https://github.com/marbl/metAMOS/archive/v1.5rc2 > v1.5rc2.zip

$ unzip v1.5rc2.zip

$ cd metAMOS-v1.5rc2

$ python INSTALL.py iMetAMOS
```

To enable iMetAMOS, specify it as an option to initPipelien using the -W flag. Below is a simple example of running of iMetAMOS to assemble an SRA dataset:

```
initPipeline -q -1 SRR987657 -d projectDir -W iMetAMOS
runPipeline -d projectDir -p 16
```

Workflows

Workflows (and common use cases)

What is a workflow?

Good question! A workflow is a text-file that specified command-line options and input sequences required to run metAMOS. A workflow may optionally inherit options/data from other workflows. A workflow may also be immutable if the parameters should not be modifiable by a user.

Example workflow

An example workflow:

```
inherit:            isolate
modify:             True
command:            -q -u -r -v -I -c kraken -p 16 -a spades,velvet-sc,abyss,ray,
↪edena,sga,masurca,soap,soap2,velvet -t metamos -n FunctionalAnnotation -f␣
↪Postprocess -z phylum
asmcontigs:         /Users/skoren/Personal/Research/metAMOS/Test/test.asm,ftp://ftp.
↪ncbi.nih.gov/genomes/Bacteria/Candidatus_Carsonella_ruddii_uid58773/NC_008512.fna
lib1format:         fasta
lib1mated:          True
lib1innie:          True
lib1interleaved:    True
lib1f1:             /Users/skoren/Personal/Research/metAMOS/Test/carsonella_pe_filt.
↪fna.gz,2000,5000,3500,500
```

Available options

The available options are:

- inherit - any other workflows to inherit from. In this case, the workflow inherits options from the isolate workflow
- modify - whether users are allowed to specify command-line parameters at runtime. If false, command-line options are ignored
- command - command-line options to specify for runPipeline
- asmcontigs - optional, pre-assembled contigs to include in analysis. Can be remote file. Multiple files can be separated using commas.
- lib#format - input type for lib #. Can be fasta/fastq/sff
- lib#mated - whether the library is mated or not
- lib#innie - whether the mates are in the innie (Illumina paired-end) format or not (Illumina mate-pair)
- lib#interleaved - whether the input sequences are in a single file or in two separate files
- lib#f1 - the name of the input file, along with library min, max, mean, stdev

An arbitrary number of libraries may be specified in the above format. The below example shows an unmated library:

```
lib1format:         fasta
lib1mated:          False
lib1innie:          False
lib1interleaved:    False
lib1frg:            /Users/skoren/Personal/Research/metAMOS/Test/carsonella_pe_filt.
↪fna.gz
```

as well as a non-interleaved library:

```
lib1format:         fasta
lib1mated:          True
lib1innie:          True
lib1interleaved:    False
lib1f1:             /Users/skoren/Personal/Research/metAMOS/Test/carsonella_pe_1.fna.
↪gz,2000,5000,3500,500
lib1f2:             /Users/skoren/Personal/Research/metAMOS/Test/carsonella_pe.2.fna.
↪gz,2000,5000,3500,500
```

Sharing your favorite MA workflows with others

Workflows may be shared between users, as long as the input files are accessible (i.e. they are on a remote server or the systems share a file system). Workflow files should be placed in the metAMOS/workflows directory or the working directory where MetAMOS is launched.

Generic tools (or plug-in framework)

Description

MetAMOS allows new tools to be added to the ASSEMBLE and ANNOTATE steps without requiring code changes.

Contributing to metAMOS

If you add an assembler or classifier or have alternate assembler parameters that you believe will benefit the community, please post the required spec file and citation either as a new issue through a pull request.

How-to-use

The addition of a tool is a three (or four) step process; we will now review the required four steps.

Add the tool name under metAMOS/Utilities/<STEPNAME>.generic. For example. if you want to add a new assembler, you would modify ASSEMBLE.generic. This file contains one tool name per line. The tool name is arbitrary text and will be used by MetAMOS to look up detailed configuration. The current ASSEMBLE.generic looks like:

```
>cat Utilities/config/ASSEMBLE.generic

abyss
sga
spades
ray
masurca
mira
edena
idba-ud
```

Note: You can add multiple versions of an assembler. In this documentation, we will add SOAPdenovo v1.05 in addition to the above tools. First, we will add soap_v105 to the end of ASSEMBLE.generic:

```
> cat Utilities/config/ASSEMBLE.generic
abyss
sga
spades
ray
masurca
mira
edena
idba-ud
soap_v105
```

The configuration file specifies input requirements for the program as well as a name, output, and executable location. Within configuration files, several keywords may be specified that are updated at runtime. The list of currently sup-

ported keywords can be found at the end of this section. In the above example, MetAMOS would expect a file named soap_v105.spec.

Below is an example configuration file used for Ray:

```
 > cat Utilities/config/ray.spec

 [CONFIG]
 maxlibs 1
 input FASTQ
 name Ray
 output [PREFIX]_ray/Contigs.fasta
 scaffoldOutput [PREFIX]_ray/Scaffolds.fasta
 location cpp/[MACHINE]/Ray/bin
 threads -n
 paired_interleaved -i [FIRST]
 paired -p [FIRST] [SECOND]
 commands rm -rf [RUNDIR]/ray && [MPI] [THREADS] Ray -o [RUNDIR]/[PREFIX]_ray [INPUT]
 unpaired -s [FIRST]
[Ray]
k
[KMER]
```

The [CONFIG] section is the generic configuration section, you can specify step-specific configuration later on. Here, most properties of where the tool is located, what its output is, and what input it requires is specified:

- input - the type of input (FASTQ in this case)
- name - the full name of the tool you want to report later on. This can be arbitrary text.
- output - where the output contigs from the tool are. For assemblers, this is contigs. [PREFIX] is a keyword for the MetAMOS prefix for the assembly when it is run. This is assumed to be relative to the MetAMOS run directory.
- scaffoldOutput - where the output scaffolds from the tool are, if available.
- backupOutput - some assemblers fail to generate their final output on some datasets. In this case, this can specify preliminary contig output which will only be used if the main output is not available.
- location - path to the executable. This is relative to metAMOS/Utilities. You can specify [MACHINE] to substitute your machine type into the executable path (i.e. Linux-x86_64). The user path will be searched if the tool is not found in the specified location
- threads - the parameter to pass number of threads to use for the program, if available
- paired - how to pass paired-end (assumed innie) interleaved data (FIRST refers to left mates, SECOND to right)
- paired_interleaved - how to pass paired-end (assumed innie) non-interleaved. FIRST refers to the interleaved file.
- mated - how to pass mate-pair data (assumed outtie) non-interleaved data (FIRST refers to left mates, SECOND to right)
- mated_interleaved - how to pass mate-pair data (assumed outtie) interleaved mates
- unpaired - how to pass fragment data to the program. FIRST refers to the unmated file.
- **commands - a list of commands to run to execute the tool. Multiple lines are supported with the character. Multiple comm**
  - [PREFIX] - the prefix to use for output
  - [RUNDIR] where the program is running

- [KMER] - the selected k-mer to use for assembly
- [MEM] - available memory
- [THREADS] - the threads parameter and number of threads requested by the user
- [INPUT] - the formatted input based on the libraries provided to metAMOS

The [Ray] section is a step-specific configuration. This is based on the executable names used in commands above. By default the parameters will be passed with prefixed - so here Ray will be run with -k [KMER]

Some assemblers (SOAPdenovo, MaSuRCA, etc) require an input configuration file rather than taking parameters on the command line. In this case, we need both a spec and template file (soap_v105.spec and soap_v105.template) which will get updated at runtime and passed to the assembler. The [CONFIG] section then includes a config option which specifies the template and the keyword [INPUT] will pass the configuration file rather than library information.

Below is an example spec file for SOAPdenovo that requires a template and spec file:

```
>cat Utilities/config/soap_v105.spec
[CONFIG]
input FASTQ
name soap_v105
threads -p
output [PREFIX]/[PREFIX].asm.contig
location cpp/[MACHINE]/SOAPdenovo_1.05/
scaffoldOutput [PREFIX]/[PREFIX].asm.scafSeq
config config/soap_v105.template
mated rank=[LIB]\navg_ins=[MEAN]\nreverse_seq=1\nasm_flags=2\nq1=[FIRST]\nq2=[SECOND]
paired rank=[LIB]\navg_ins=[MEAN]\nreverse_seq=0\nasm_flags=3\nq1=[FIRST]\nq2=[SECOND]
unpaired rank=[LIB]\navg_ins=0\nq=[FIRST]
commands rm -rf [PREFIX] && mkdir [PREFIX] && SOAPdenovo all -s [INPUT] -o [PREFIX]/
↪[PREFIX].asm -K [KMER] [THREADS]

>cat Utilities/config/soap_v105.template
#maximal read length
max_rd_len=150
[LIB]
[INPUT]
```

Here, the config template is specified (again relative to metAMOS/Utilities) and the [INPUT] keyword will be replaced by the library information at run time.

Citations are tab-delimited and specify the lower-case tool alias, full tool-name, and citation information. For example:

```
soap_v105   SOAPdenovo v1.05        Li Y, Hu Y, Bolund L, Wang J: State of the art de␣
↪novo assembly of human genomes from massively parallel sequencing data.Human␣
↪genomics 2010, 4:271-277.
```

The citation will be automatically printed by MetAMOS whenever a run uses the specified tool.

For ANNOTATE tools, we also need a way to convert the output to Krona. By default, MetAMOS will look for an Import<toolName>.pl script. If one is not found, it will rely on a generic import which will assumed a tab-delimited format:

```
contig/readID       NCBI Taxonomy ID
```

The currently supported list of keywords:

- MEM - max memory limit
- LIB - library identifier (i.e. 1, 2, 3, etc)

- INPUT - replace with input to the program (a collection of input files or libraries depending on the step or a configuration file)
- MACHINE - replaced with Linux-x86_64, Darwin-x86_64, etc
- FIRST - replaced with left mates in mated read or interleaved or unpaired reads otherwise
- SECOND - replaced with right mates, in paired non-interleaved libs
- ORIENTATION - replaced with the word innie or outtie
- ORIENTATION_FIGURE - replaced with —> <— or <— —> for pe and mp, respectively
- MEAN - replaced with library mean
- SD - replaced with library standard dev
- MIN - replaced with library min
- MAX - replaced with library max
- THREADS - replaced with thread parameter specified and requested number of threads
- KMER - the kmer requested
- OFFSET - the phred offset (33/64) of the input files
- PREFIX - the desired prefix for the program output
- DB - the location of the MetAMOS DBs (i.e. Utilities/DB)
- RUNDIR - the location where the program is running (i.e. MetAMOS run directory)
- LOCATION - the location where the program executable lives
- TECHNOLOGY - the type of sequencing data (454, Illumina, etc)

MetAMOS directory structure

Directory layout/description

All of the step mentioned below have the following directory structure:

```
[STEP]/in  -> required input
[STEP]/out -> generated output
```

We will now describe in detail the functionality of each step, along with the expected input & output.

Preprocess

Required step?

- Yes

Software currently supported

- ea-utils (code.google.com/p/ea-utils) - optional, off by default. Enabled by passing and trim option to runPipeline.

```
$ -t eautils
```

  - Assemblers that do not perform trimming can benefit from enabling this step. On a GAGE-B dataset, the assemblers which had a higher corrected N50 on trimmed data than untrimmed were:

```
IDBA-UD, SGA, SparseAssembler, SPAdes, Velvet-SC, and Velvet.
```

  - Assembler which had a higher corrected N50 on untrimmed data were:

```
ABySS, MaSuRCA, MIRA, Ray, and SOAPdenovo2.
```

- FastQC (bioinformatics.babraham.ac.uk) - optional, on by default for iMetAMOS, used to generate quality reports for the input sequencing data.
- KmerGenie (Chikhi et al 2014) - optional, on by default for iMetAMOS, used to auto-select a k-mer for isolate genome assembly. Alternatively, a list of k-mers can be specified instead. For assemblers using a range of k-mers (i.e. IDBA-UD), KmerGenie is not used but the read length is specified as the maximum k-mer. For assemblers using a set of k-mers (i.e. SPAdes), the KmerGenie selected k-mer along with a set of defaults is used.

What it does

- Quality control
- Read filtering
- Read trimming
- Sanity checks on fasta/q files
- Conversion to required formats

Expected input

- Raw reads

Expected output

- Cleaned reads
- Quality report
- Converted files

Assemble

Required step?

- No

Software currently supported

- ABySS (Simpson et al 2009)
- CABOG (Miller et al 2008)
- IDBA-UD (Peng et al 2012)
- MaSuRCA (Zimin et al 2013)
- MetaVelvet (Namiki et al 2011)
- Mira (Chevreux et al 1999)
- RayMeta (Boisvert et al 2012)
- SGA (Simpson et al 2012)
- SOAPdenovo2 (Luo et al 2012)
- SPAdes (Bankevich et al 2012)
- SparseAssembler (Ye et al 2012)
- Velvet (Zerbino et al 2008)
- Velvet-SC (Chitsaz et al 2011)

What it does

- Construct assembly (no scaffolds)

Expected input

- Cleaned reads

Expected output

- Unitigs
- Contigs
- Singletons
- Degenerates/Surrogates

FindORFs

Required step?

- No

Software currently supported

- FragGeneScan (Rho, 2010)
- MetaGeneMark (Zhu, 2010)
- Prokka (Seemann, 2013)

What it does

- Finds/predicts ORFs in contigs

Expected input

- Assembled contigs in fasta format (>300bp)

Expected output

- ORFs in multi-fasta format (FAA,FNA)

Validate

Required step?

- No

Software currently supported

- ALE (Clark et al 2013)
- CGAL (Rahman et al 2013)
- FRCbam (Vezzi et al 2013)
- FreeBayes (Garrison et al 2012)
- LAP (Ghodsi et al 2013)
- QUAST (Gurevich et al 2013)
- REAPR (Hunt et al 2013)

What it does

- Checks assembly correctness using intrinsic quality metrics

Expected input

- Assembled contigs in fasta format

Expected output

- List of errors
- Poorly assembled regions
- Assembly quality metrics

FindRepeats (deprecated)

This step was initially added to help speed up Bambus 2 repeat identification step; optimizations to Bambus 2 have made this speed-up unnecessary. Step is turned off by default.

Required step?

- No

Software currently supported

- Repeatoire

What it does

- Find contigs (or parts of contigs) that appear to be repetitive and flag for further steps.

Expected input

- Assembled contigs in fasta format

Expected output

- List of contigs likely to be repeats

Abundance (deprecated)

This step was created to estimate taxonomic abundance of a give metagenomic sample

Required step?

- No

Software currently supported

- MetaPhyler (Liu et al 2011)

What it does

- Find contigs (or parts of contigs) that appear to be repetitive and flag for further steps.

Expected input

- Assembled contigs in fasta format

Expected output

- List of contigs likely to be repeats

Classify

Required step?

- Yes

Software currently supported

- FCP
- Kraken
- Phylosift

What it does

- Labels contigs with taxonomic id

Expected input

- Multi-fasta file of contigs

Expected output

- Text file containing contig id to taxonomic id 1-to-1 mapping

FunctionalAnnotation

Required step?

- No

Software currently supported

- BLAST

What it does

- Assigns functional annotation to ORFs

Expected input

- ORFs in multi-fasta format (FAA,FNA)

Expected output

- Text file containing functional labels for ORFs

Scaffold

Required step?

- Yes

Software currently supported

- Bambus2 (Koren, 2011)

What it does

- Link together contigs using mate-pairs. Also identify variant patterns.

Expected input

- Assembled contigs in fasta format

Expected output

- scaffolds in agp format
- scaffolds in fasta format
- motifs/variants
- longer contigs in fasta format

Propagate

Required step?

- Yes

Software currently supported

- NA

What it does

- Propagate taxonomic labels along scaffolds

Expected input

- Scaffolds in agp format
- Contig taxonomic labels

Expected output

- contig taxonomic labels

FindScaffoldORFs

Required step?

- No

Software currently supported

- FragGeneScan
- MetaGeneMark

What it does

- Find ORFs in scaffolds, mainly serves as an extra validation step after Scaffold.

Expected input

- Scaffolds in agp format

Expected output

- Multi-fasta file of ORFs as fna,faa

Binning

Required step?

- Yes

Software currently supported

- NA

What it does

- Bins contigs/scaffold by taxonomic label

Expected input

- Multi-fasta file of contigs
- Multi-fasta file of scaffolds

Expected output

- Binned out contigs/scaffolds by directory

Postprocess

Required step?

- Yes

Software currently supported

- Krona (Ondov, 2010)

What it does

- Generates summary reports
- Collates output
- Generates combined HTML page

Expected input

- Majority of the aforementioned outputs

Expected output

- HTML summary file
- Output directory tree

Output

Full listing of expected output files

MetAMOS generates an interactive web page once a run successfully completes:

```
http://www.cbcb.umd.edu/~sergek/imetamos/gageb/Postprocess/out/html/summary.html
```

This includes summary statistics and taxonomic information based on Krona [1]. The easiest way to interact with the results is through the web interface. The web interface has been tested in several browsers. The currently known issues are:

Browser	Version	Issues
Chrome	33.0.1750.152	None
Safari	6.1.2	None
Firefox	28	QUAST reports do not show for Validate
IE	9	Not Tested

The Postprocess/out directory contains the results of the analysis. By default, metAMOS uses the prefix "proba" (Galician for test). Thus, files will have the name "proba".*.

abundance.krona.html

Krona [1] plot of abundances using the tool selected for abundance (MetaPhyler [2] by default)

annotate.krona.html

Krona [1] plot of abundances using the tool selected for classification (Kraken [3] by default)

asm.scores

Validation scores for each assembly/kmer combination run. Header contains information on scores generated

best.asm

The name of the assembly/kmer combination that was selected as the best

<taxonomy>.classified

Subdirectory containing each level of the selected taxonomy (class by default) and the contigs/reads/orfs belonging to each

<taxonomy>.original.annots

Tab-delimited taxonomic level assignments for each contig/unassembled read. Class IDs correspond to NCBI taxonomy IDs.

<taxonomy>.original.reads.annots

Tab-delimited taxonomic level assignments as above, where contigs are replaced with their constituent sequences.

<taxonomy>.propagated.annots

Tab-delimited file as above after assembly graph-based propagation of assignments to contigs.

<taxonomy>.propagated.reads.annots

Tab-delimited file as above after propagation and having contigs replaced with their constituent reads.

html (directory)

HTML output from the pipeline. summary.html contains an interactive results view.

proba.bnk

AMOS bank format of the assembly that can be visualized using Hawkeye.

proba.classify.txt

The raw output of the abundances using the tool selected for abundance estimations (MetaPhyler [2] by default)

proba.ctg.cnt

The number of sequences mapped to each assembly contig

proba.ctg.cvg

The coverage of each assembly contig

proba.ctg.fa

The assembled contigs

proba.hits

The raw output of the contig/unassembled reads classifications using the selected tool (Kraken [3]) by default.

proba.lib1.contig.reads

The per-library assignment of sequences to contigs

proba.lib1.unaligned.fasta

The per-library unassembled sequences

proba.scf.fa

The assembled scaffolds

proba.motifs.fa

The motifs within scaffolds identified by Bambus 2

proba.orf.faa

The protein sequences of identified open reading frames (ORFs) in the assembly and unassembled reads

proba.orf.fna

The fasta sequences of identified open reading frames (ORFs) in the assembly and unassembled reads

proba.scf.orf.faa

The protein sequences of identified open reading frames (ORFs) in the scaffolds

proba.scf.orf.fna

The protein sequences of identified open reading frames (ORFs) in the scaffolds

ref.fasta

The recruited reference genome used for validation (iMetAMOS only)

ref.name

The name of the recruited reference genome (iMetAMOS only)

Additional details for each step are available under <STEP NAME>/out. This includes the raw output (as well as any intermediate files) of any tools run during that step. For example, Annotate/out/proba.prokka includes the full Prokka annotation output. Assemble/out/abyss*/ contains the intermediate files output by ABySS. Additionally, since MetAMOS stores all of its results in an AMOS bank, the assemblies can be visualized with Hawkeye.

[1] Ondov BD, Bergman NH, Phillippy AM.. Interactive metagenomic visualization in a Web browser. BMC Bioinformatics. 2011 Sep 30;12:385. PMID: 21961884

[2] Liu B, Gibbons T, Ghodsi M, Treangen T, Pop M. Accurate and fast estimation of taxonomic profiles from metagenomic shotgun sequences. BMC Genomics. 2011;12 Suppl 2:S4. Epub 2011 Jul 27.

[3] Wood DE, Salzberg SL: Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biology 2014, 15:R46.

Supported Programs

MetAMOS depends on many publically-available software tools. Below is a list of currently supported programs along with their citations:

Preprocess/Filtering

EA-UTILS: Aronesty E. TOBioiJ : "Comparison of Sequencing Utility Programs", DOI:10.2174/1875036201307010001, 2013.

PBcR: Koren S, Harhay GP, Smith TPL, Bono JL, Harhay DM, Mcvey SD, Radune D, Bergman NH, Phillippy AM. Reducing assembly complexity of microbial genomes with single-molecule sequencing. Genome Biology 14:R101 2013.

KmerGenie: Chikhi, R, Medvedev, P. Informed and Automated k-Mer Size Selection for Genome Assembly. Bioinformatics btt310, 2013.

Assemblers

SPAdes: Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. Journal of Computational Biology. May 2012, 19(5): 455-477. doi:10.1089/cmb.2012.0021.

Edena: Hernandez D, Tewhey R, Veyrieras J, Farinelli L, Østerås M, François P, and Schrenzel J. De novo finished 2.8 Mbp Staphylococcus aureus genome assembly from 100 bp short and long range paired-end reads. Bioinformatics, btt590, 2013.

SOAPdenovo: Li Y, Hu Y, Bolund L, Wang J: State of the art de novo assembly of human genomes from massively parallel sequencing data.Human genomics 2010, 4:271-277.

SOAPdenovo2: Luo, R, Liu, B, Xie, Y, Li, Z, Huang, W, Yuan, J, He G, Chen Y, Pan Q, Liu Y, Tang J, Wu G, Zhang H, Shi Y, Liu Y, Yu C, Wang B, Lu Y, Han C, Cheung DW, Yiu S, Peng S, Xiaoqian Z, Liu G, Liao X, Li Y, Yang H, Wang J, Lam T, Wang J. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. GigaScience, 1(1), 18, 2012.

IDBA-UD: Peng, Y., Leung, H. C., Yiu, S. M., & Chin, F. Y. IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. Bioinformatics, 28(11), 1420-1428, 2012.

Meta-IDBA: Peng Y, Leung HCM, Yiu SM, Chin FYL: Meta-IDBA: a de Novo assembler for metagenomic data. Bioinformatics 2011, 27:i94-i101.

Velvet: Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res. 2008 May;18(5):821-9.

MetaVelvet: Namiki, T., Hachiya, T., Tanaka, H., & Sakakibara, Y. MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads. Nucleic acids research, 40(20), e155-e155, 2012.

Celera Assembler: Miller JR, Delcher AL, Koren S, Venter E, Walenz BP, Brownley A, Johnson J, Li K, Mobarry C, Sutton G. Aggressive assembly of pyrosequencing reads with mates.Bioinformatics. 2008 Dec 15;24(24):2818-24. Epub 2008 Oct 24.

Minimus: Sommer DD, Delcher AL, Salzberg SL, Pop M. Minimus: a fast, lightweight genome assembler. BMC Bioinformatics. 2007 Feb 26;8:64.

Sparse Assembler: Ye C, Ma ZS, Cannon CH, Pop M, Yu DW. Exploiting sparseness in de novo genome assembly. BMC Bioinformatics. 2012 Apr 19;13 Suppl 6:S1.

Velvet-SC: Chitsaz H, Yee-Greenbaum JL, Tesler G, Lombardo MJ, Dupont CL, Badger JH, Novotny M, Rusch DB, Fraser LJ, Gormley NA, Schulz-Trieglaff O, Smith GP, Evers DJ, Pevzner PA, Lasken RL. Efficient de novo assembly of single-cell bacterial genomes from short-read data sets. Nature Biotechnology, vol. 29, no. 11, pp. 915-921 (2011)

MaSuRCA: Zimin, A, Marçais, G, Puiu, D, Roberts, M, Salzberg, SL, Yorke, JA. The MaSuRCA genome assembler. Bioinformatics, btt476, 2013.

Ray: Boisvert, S, Raymond, F, Godzaridis, É, Laviolette, F, Corbeil, J. Ray Meta: scalable de novo metagenome assembly and profiling. Genome biology, 13(12), R122, 2013.

ABySS: Simpson, JT, Wong, K, Jackman, SD, Schein, JE, Jones, SJ, Birol, İ. ABySS: a parallel assembler for short read sequence data. Genome research, 19(6), 1117-1123, 2009.

SGA: Simpson, JT, Durbin, R. Efficient de novo assembly of large genomes using compressed data structures. Genome Research, 22(3), 549-556, 2012.

MIRA: Chevreux, B, Wetter, T, Suhai, S. Genome Sequence Assembly Using Trace Signals and Additional Sequence Information. In German Conference on Bioinformatics (pp. 45-56), 1999.

Read Mapping

Bowtie: Langmead B, Trapnell C, Pop M, Salzberg SL. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. 2009;10(3):R25. Epub 2009 Mar 4.

Bowtie2: Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nat Methods. 2012 Mar 4;9(4):357-9. doi: 10.1038/nmeth.1923.

Classifier

FCP,Naive Bayesian Classifier: Macdonald NJ, Parks DH, Beiko RG. Rapid identification of high-confidence taxonomic assignments for metagenomic data. Nucleic Acids Res. 2012 Apr 24.

BLAST: Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990 Oct 5;215(3):403-10.

PHMMER: Eddy SR. Accelerated Profile HMM Searches. PLoS Comput Biol. 2011 Oct;7(10):e1002195. Epub 2011 Oct 20.

PHYMM: Brady A, Salzberg SL. PhymmBL expanded: confidence scores, custom databases, parallelization and more. Nat Methods. 2011 May;8(5):367.

PhyloSift: Darling, AE, Jospin, G, Lowe, E, Matsen IV, FA, Bik, HM, Eisen, JA. PhyloSift: phylogenetic analysis of genomes and metagenomes. PeerJ, 2, e243, 2014.

MetaPhyler: Liu B, Gibbons T, Ghodsi M, Treangen T, Pop M. Accurate and fast estimation of taxonomic profiles from metagenomic shotgun sequences. BMC Genomics. 2011;12 Suppl 2:S4. Epub 2011 Jul 27.

Kraken: Wood DE, Salzberg SL: Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biology 2014, 15:R46.

Annotation/GeneFinding

FragGeneScan: Rho M, Tang H, Ye Y: FragGeneScan: predicting genes in short and error-prone reads. Nucleic Acids Research 2010, 38:e191-e191.

MetaGeneMark: Borodovsky M, Mills R, Besemer J, Lomsadze A: Prokaryotic gene prediction using GeneMark and GeneMark.hmm.Current protocols in bioinformatics editoral board Andreas D Baxevanis et al 2003, Chapter 4:Unit4.6-Unit4.6.

Prokka: Prokka: Prokaryotic Genome Annotation System - http://vicbioinformatics.com/

Glimmer-MG: Kelley DR, Liu B, Delcher AL, Pop M, Salzberg SL. Gene prediction with Glimmer for metagenomic sequences augmented by classification and clustering. Nucleic Acids Res. 2012 Jan;40(1):e9. Epub 2011 Nov 18.

Validation

LAP: Ghodsi M, Hill CM, Astrovskaya I, Lin H, Sommer DD, Koren S, Pop M. De novo likelihood-based measures for comparing genome assemblies. BMC research notes 6:334, 2013.

ALE: Clark, SC, Egan, R, Frazier, PI, Wang, Z. ALE: a generic assembly likelihood evaluation framework for assessing

the accuracy of genome and metagenome assemblies. Bioinformatics, 29(4) 435-443, 2013.

QUAST: Gurevich, A, Saveliev, V, Vyahhi, N, Tesler, G. QUAST: quality assessment tool for genome assemblies. Bioinformatics, 29(8), 1072-1075, 2013.

FRCbam: Vezzi, F, Narzisi, G, Mishra, B. Reevaluating assembly evaluations with feature response curves: GAGE and assemblathons. PloS ONE, 7(12), e52210, 2013.

CGAL: Rahman, A, Pachter, L CGAL: computing genome assembly likelihoods. Genome biology, 14(1), R8, 2013.

FreeBayes: Garrison, E, Marth, G. Haplotype-based variant detection from short-read sequencing. arXiv preprint arXiv:1207.3907, 2012.

REAPR: Hunt, M, Kikuchi, T, Sanders, M, Newbold, C, Berriman, M, & Otto, TD. REAPR: a universal tool for genome assembly evaluation. Genome biology, 14(5), R47, 2013.

Scaffolders

Bambus 2: Koren S, Treangen TJ, Pop M. Bambus 2: scaffolding metagenomes. Bioinformatics 27(21): 2964-2971 2011.

Miscelaneous

M-GCAT: Treangen TJ, Messeguer X. M-GCAT: interactively and efficiently constructing large-scale multiple genome comparison frameworks in closely related species. BMC Bioinformatics, 2006.

SAMtools: Li H., Handsaker B.*, Wysoker A., Fennell T., Ruan J., Homer N., Marth G., Abecasis G., Durbin R. and 1000 Genome Project Data Processing Subgroup (2009) The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics, 25, 2078-9

Krona: Ondov BD, Bergman NH, Phillippy AM. Interactive metagenomic visualization in a Web browser. BMC Bioinformatics. 2011 Sep 30;12:385.

FAQs

Q: How should I install MetAMOS?

A: We recommend using the provided INSTALL.py script, that will retrieve and compile are requisite dependencies. As a shortcut we provide a PyInstaller-powered frozen binary; this is primarily for users who are experiencing extreme difficulties installing via INSTALL.py.

Q: Where are my $#%^! results ?!?

A: See here and here. If you still have questions, contact the dev team.

Q: Why is the FindORFs step taking so long to complete?

A: FragGeneScan is the default metagenomic gene caller; to improve performance we suggest acquiring a license to incorporate MetaGeneMark into your MetAMOS install.

Q: What steps can I skip?

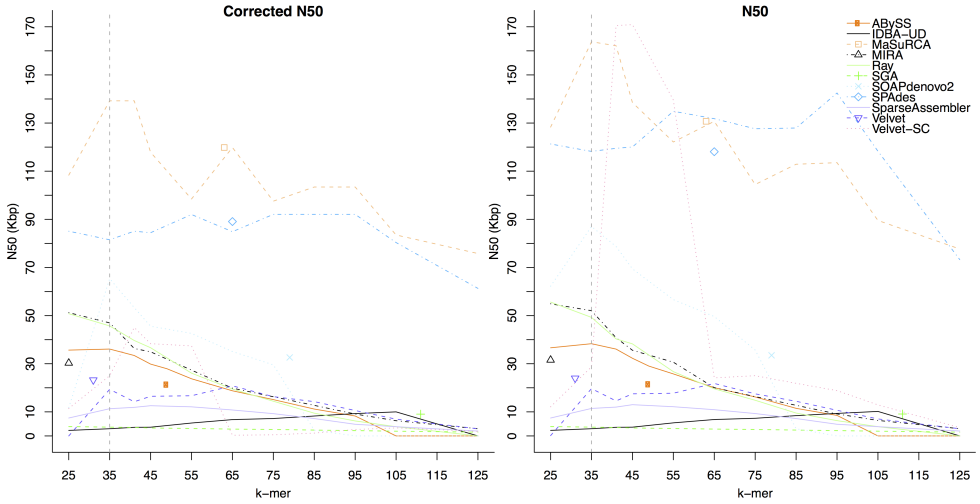
A: Most of them! required steps are currently: Preprocess, Scaffold, and Postprocess.

Q: Should I trim my input data?

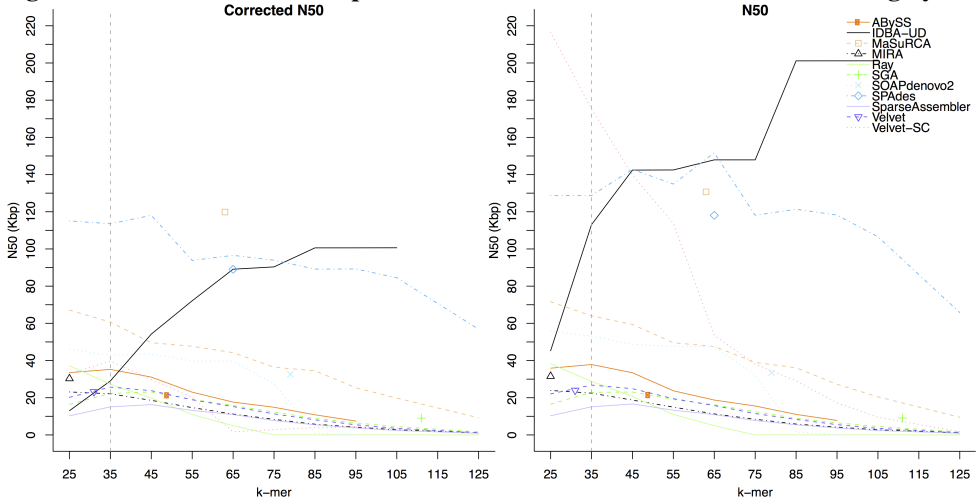
A: iMetAMOS supports EA-UTILS as a trimming option, though it is disabled by default. We have found that some assemblers that build-in their own trimming module are hampered by pre-trimming the data. In contrast, assembler that do not have a trimming module can benefit from trimming input sequences. To enable trimming using EA-UTILS, a trimming option can be specified to runPipeline

```
$ -t eautils
```

- **We compared assembler performance on trimmed and untrimmed data for the GAGE-B MiSeq Rhodobacter sp**



- **The figure below shows assembler performance on the same dataset after trimming by EA-UTILS**



- On this dataset, the assemblers which had a higher corrected N50 on trimmed data than untrimmed were:

```
IDBA-UD, SGA, SparseAssembler, SPAdes, Velvet-SC, and Velvet.
```

- Assembler which had a higher corrected N50 on untrimmed data were:

```
ABySS, MaSuRCA, MIRA, Ray, and SOAPdenovo2.
```

Q: What taxonomic classification method should I be using?

A: Good question! But in our experience there is no single method to universally recommend. If you'd like a ultrafast method with great precision but are less worried about sensitivity, Kraken performs well. If you are less concerned about assigning labels to contigs/reads and would rather like to phylogenetically place your reads/contigs w.r.t marker genes, PhyloSift is recommended.

Q: Help! The frozen binary will not extract or unexpectedly crashes.

A: The most common reason for this occuring is a lack of free space in the /tmp directory. So first double check that the temporary directory has sufficient space and permissions for the current user. By default, PyInstaller will search a standard list of directories and sets tempdir to the first one which the calling user can create files in. On most systems this will be:

```
$/tmp/_MEI*
```

The list is:

- The directory named by the TMPDIR environment variable.
- The directory named by the TEMP environment variable.
- The directory named by the TMP environment variable.

  If your system is missing all of the above, or all of the directories have insufficient free space, runPipeline will not be able to extract itself and will fail while running (see github issue #121)

Contact

Bugs, feature requests, comments:

If you encounter any problems/bugs, please check the known issues pages:

```
https://github.com/treangen/MetAMOS/issues?direction=desc&sort=created&state=open
```

If not, please report the issue either using the contact information below or by submitting a new issue online.

Please include information on your run:

```
1) any output produced by runPipeline

2) the pipeline.* files

3) Log/<LAST_STEP> file (if not too large).
```

Who to contact to report bugs, forward complaints, feature requests:

Sergey Koren: sergek@umd.edu

Todd J. Treangen: treangen@gmail.com

Experimental: TweetAssembler v0.1b

Introduction

TweetAssembler is a twitter-based interface to an isolate genome assembly server powered by iMetAMOS:

```
Automated ensemble assembly and validation of microbial genomes.
Sergey Koren, Todd J Treangen, Christopher M Hill, Mihai Pop, Adam M Phillippy
BMC Bioinformatics 15:126, 2014.
http://www.biomedcentral.com/1471-2105/15/126/abstract
```

Why Twitter?

- Good question! The main Raison d'être of TweetAssembler is to highlight the utility of iMetAMOS; just point it to your reads (no other params required!) and it will preprocess, tune, assemble, validate and create an HTML report of the results. This enables the submit command to be readily constructed in fewer than 140 chars.

Limitations

Before proceeding, its important to higlight a few important points:

- The server behind TweetAssembler is only able to assemble a couple of requests (at best) per day. Specs are: 32GB RAM & 32GhZ of compute... be gentle!
- There exists limitations on the size of the input data. i.e. MiSeq ok, HiSeq not ok.
- TweetAssembler is nothing more than a tweet-based interface to an iMetAMOS webserver.
- Given the limited resources, job queue management is disabled. You will only be able to run a job if no other jobs are active; your only indication that your job was accepted is the confirmation tweet (see below).
- Twitter has a maximum # of tweets allowed per day (1000), as well hourly limits. If TweetAssembler goes over any of these limits it will be deactivated for approx. 1 hour, potentially longer.
- No guarantees on preservation of output! Assemblies & associated output can & will be deleted regularly.

Quick Start

1. First, issue a request to follow @imetamos:



2. Next, contact the developers to get your twitter account added to the *allowed accounts* list:

- Todd J Treangen (treangen@gmail.com)

- Sergey Koren (sergekoren@gmail.com)

3. Once approved, compose a tweet to @imetamos using the following syntax:

```
@imetamos [fastq_pair_1] [fastq_pair_2] [#ASSEMBLE] [id]
```

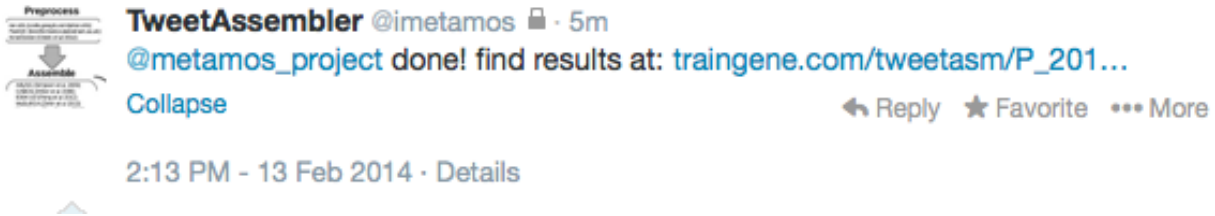
- Think of an automated #icanhazpdf but for genome assemblies (#icanhazasm).
- Currently reads need to be in non-interleaved fastq format.
- id simply needs to be a job-unique integer to avoid duplicate tweets in case you have to submit your job multiple times before it runs.
- You should notice that no parameters are required (except for the input data). In practice this works thanks to several software packages, e.g. kmergenie (http://kmergenie.bx.psu.edu/), and an ensemble assembly approach (powered by several assembly and assembly validation tools).



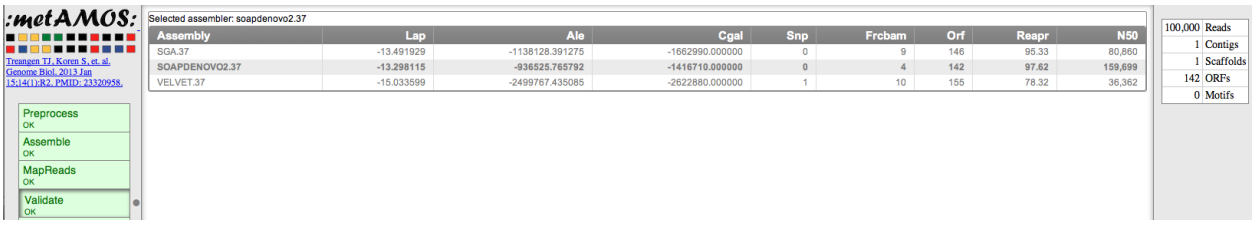
4. You should immediately receive a response tweet similar to:



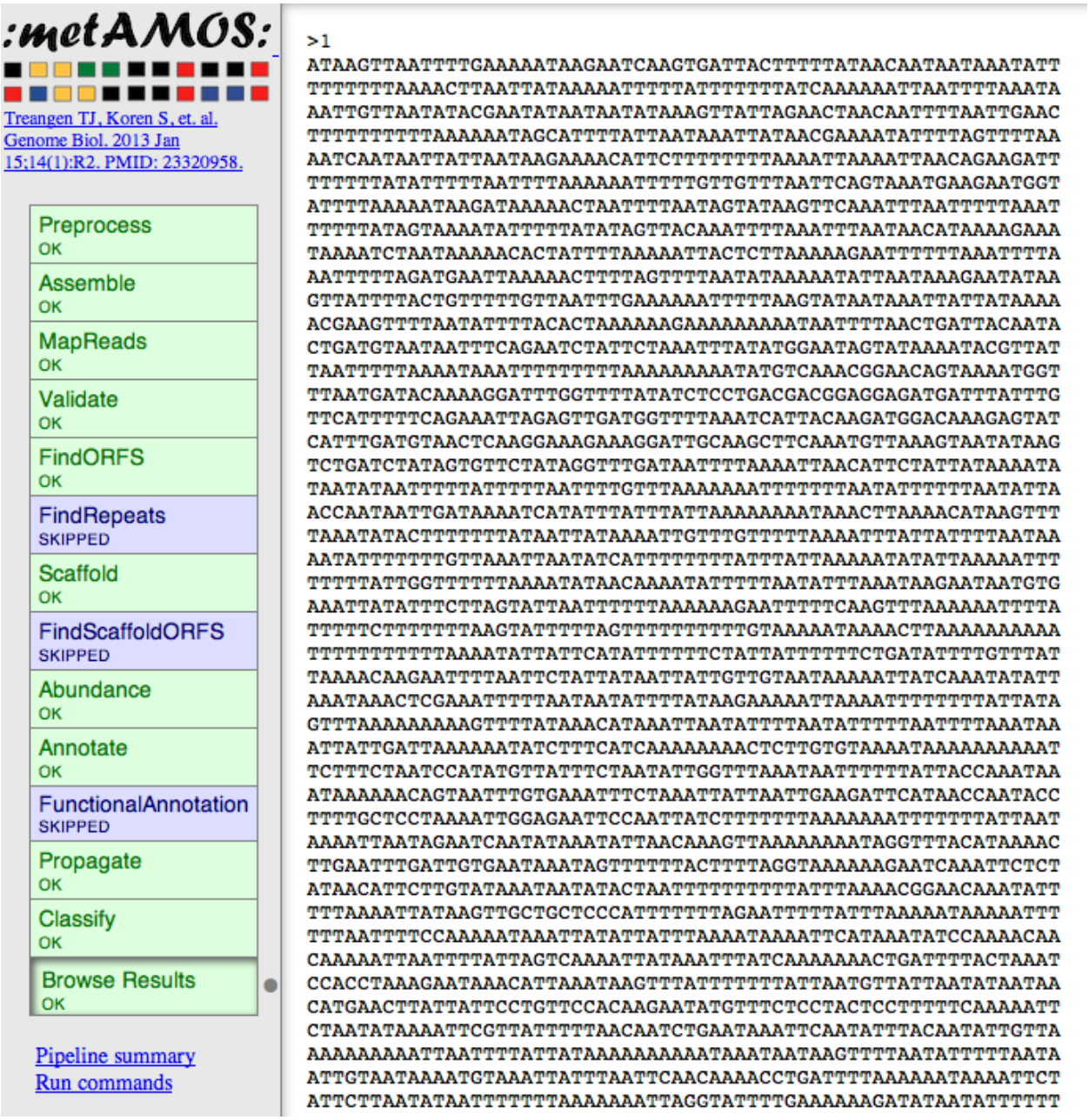
5. Then simply wait for the confirmation tweet that the job was successful.



6. Upon completion, you will be able to view the HTML report :



7. and download your assembly:



8. Suggestions & comments welcome!

Viewing Output

Your output will be located at http://www.traingene.com/tweetasm/P_{[}TIMESTAMP{]}/out/html/summary.html.

- Example output: http://www.traingene.com/tweetasm/P_2014_02_11_142926937305/out/html/summary.html

To save assembly:

```
wget http://www.traingene.com/tweetasm/P_[TIMESTAMP]/out/proba.ctg.fa
```

Supported Software

Last but not least, we would like to acknowledge all of the wonderful software that provides the firepower behind TweetAssembler and iMetAMOS:

[Preprocess]

- ea-utils (code.google.com/p/ea-utils)
- FastQC (bioinformatics.babraham.ac.uk)
- KmerGenie (Chikhi et al 2014)

[Assemble]

- ABySS (Simpson et al 2009)
- CABOG (Miller et al 2008)
- IDBA-UD (Peng et al 2012)
- MaSuRCA (Zimin et al 2013)
- MetaVelvet (Namiki et al 2011)
- Mira (Chevreux et al 1999)
- RayMeta (Boisvert et al 2012)
- SGA (Simpson et al 2012)
- SOAPdenovo2 (Luo et al 2012)
- SPAdes (Bankevich et al 2012)
- SparseAssembler (Ye et al 2012)
- Velvet (Zerbino et al 2008)
- Velvet-SC (Chitsaz et al 2011)

[MapReads]

- Bowtie (Langmead et al 2009)
- Bowtie2 (Langmead et al 2012)

[Validate]

- ALE (Clark et al 2013)
- CGAL (Rahman et al 2013)
- FRCbam (Vezzi et al 2013)
- FreeBayes (Garrison et al 2012)
- LAP (Ghodsi et al 2013)
- QUAST (Gurevich et al 2013)
- REAPR (Hunt et al 2013)

[FindORFS/Annotate]

- Prokka (Seemann, 2013)

thanks!