Contents

1 Introduction 1

2 Quick Start Guide 1
   2.1 Installation ................................................................. 1
   2.2 Running BugBuilder .................................................... 1

3 Installation 2
   3.1 Virtual Machine Installation ........................................... 2
      3.1.1 Starting the Virtual Machine .................................. 5
   3.2 Virtual Machine Post-Installation Configuration .................. 6
   3.3 Full Installation ........................................................... 6
      3.3.1 Obtaining the software ........................................... 6
   3.4 Automated Installation .................................................. 6
   3.5 Manual Installation ........................................................ 8
      3.5.1 Operating System Packages ...................................... 8
      3.5.2 Prerequisite Software .............................................. 8
      3.5.3 sam2afg ............................................................... 9
      3.5.4 Perl Modules ....................................................... 9
   3.6 Downloading Example Data Sets ...................................... 10

4 Pipeline Workflow 10
   4.1 Pipeline Inputs ............................................................ 10
   4.2 Pipeline Outputs .......................................................... 11
   4.3 Workflow ................................................................. 11
   4.4 Sequence QC ............................................................. 11
4.5 Downsampling ................................................. 12
4.6 Assembly ...................................................... 12
  4.6.1 Assembly with Multiple Assemblers .................... 12
4.7 Scaffolding .................................................... 12
4.8 Scaffold Orientation and Origin Identification .......... 13
4.9 Gap Closure .................................................... 14
4.10 AGP Generation .............................................. 14
4.11 Annotation .................................................... 14
4.12 Outputs ......................................................... 14
4.13 Output Visualisation ........................................ 15
  4.13.1 MUMmerplot ............................................. 15
  4.13.2 ACT Comparison .......................................... 15
  4.13.3 Circleator Visualisation ................................ 15
5 Submission vs Draft Modes .................................. 16
  5.1 Submission Mode ............................................ 16
  5.2 Draft Mode ................................................... 17
6 Running BugBuilder ............................................ 17
  6.1 Setting Assembler Parameters ............................. 18
  6.2 Running Multiple Assemblers ............................... 18
  6.3 Specifying Organism Details ............................... 18
  6.4 Tutorial: Running the Example Assemblies ............. 19
    6.4.1 Illumina GAII ........................................... 19
    6.4.2 Illumina MiSeq .......................................... 20
    6.4.3 454 ...................................................... 21
    6.4.4 PacBio .................................................. 21
    6.4.5 Hybrid PacBio/Illumina ................................ 22
7 Advanced Configuration ...................................... 22
  7.1 Temporary directory location .............................. 22
  7.2 Software Installation Locations ........................... 22
  7.3 Assembler Configuration .................................... 23
1 Introduction

BugBuilder is a pipeline enabling assembly and annotation of microbial genomes from high-throughput sequence data. It is designed to minimise manual work in creating annotated, draft genome assemblies ready for submission to public databases. BugBuilder offers a configurable framework capable of supporting a number of different assemblers and scaffolders which are selectable at runtime. These provide support for all assembly of data generated with all common sequencing platforms (Illumina, 454, IonTorrent, PacBio and MinIon), including hybrid assembly between e.g. Illumina and PacBio sequences. It is implemented in Perl and has been tested on Red Hat derived Linux distributions, however should work on any Unix-like operating system which supports the prerequisite software components.

BugBuilder is available from the BugBuilder home page at http://www.imperial.ac.uk/bioinf-support/software/BugBuilder The code is maintained in a GitHub repository which can be found at https://github.com/jamesabbott/BugBuilder.

2 Quick Start Guide

2.1 Installation

Installation and configuration of a standalone instance of BugBuilder requires a degree of knowledge of Linux software installation and management, and installation of a considerable number of prerequisite packages, so it is recommended to initially try out BugBuilder using the pre-configured virtual machine image (see section 3.1 Virtual Machine Installation), which requires a minimal amount of setup work.

2.2 Running BugBuilder

Once BugBuilder is available on your system, you can perform an assembly of a paired-end sequencing run simply by entering the command:

```
BugBuilder --fastq1 read1.fastq --fastq2 read2.fastq --platform illumina
```

where 'read1.fastq' contains the reads from the first read of the pair, and read2.fastq contains the second read, and the 'platform' argument defines the sequencing platform used to generate the reads. If your sequence was generated from a fragment library rather than a paired-end library, the '-fastq2' argument should be omitted. BugBuilder will proceed to carry out the assembly, reporting it's progress to the screen. Once complete, the generated outputs will be written to a new directory within the current working directory.

If you have an appropriate reference genome sequence (ideally a complete genome from the same genus, or preferably species), this can be used to guide the scaffolding process typically producing fewer scaffolds. The reference genome needs to be provided in fasta format, and can be included in the analysis using the '-reference' argument:

```
BugBuilder --fastq1 read1.fastq --fastq2 read2.fastq --reference myreference.fasta --platform illumina
```

Your BugBuilder installation may have been configured with multiple assemblers and scaffolders. Certain assemblers are appropriate, or perform better, with particular sequence types, and BugBuilder can select the most appropriate tools for your assembly based on the sequencing platform used, and the length of your
sequence reads. These appropriate tools for each platform are centrally configured within the BugBuilder configuration file.

To see which assemblers, scaffolders and platforms are configured, view the help documentation by entering:

[james@codon ~]$ BugBuilder --help

Welcome to BugBuilder

Available assemblers: SPAdes, abyss, celera
Available scaffolders: SIS, sspace, mauve
Configured platforms: 454, illumina, iontorrent, hybrid

An appropriate assembler and scaffoldor for the type of sequence available will be selected from the configured options based on the sequencing platform supplied using the ‘--platform’ argument: Alternatively, rather than using the automatically selected tools, the assembler and scaffoldor can be directly specified using the ‘--assembler’ and ‘--scaffolder’ arguments:

BugBuilder --fastq1 read1.fastq.gz --fastq2 read2.fastq.gz --platform illumina
--reference myreference.fasta --assembler spades --scaffolder SIS

3 Installation

There are a considerable number of prerequisite packages which are required for BugBuilder, and a configuration file which needs to be setup to indicate the location of the installed packages, and defines the available assemblers and scaffoldors. This can make installation a fairly complex process. A preconfigured virtual machine is therefore available which is installed and configured with all freely distributable packages and a script to automate the installation of the remaining software which it is not possible to distribute directly due to restrictive licenses. The virtual machine image should provide a functional environment where BugBuilder can be run, however those making heavy use of the software would benefit from carrying out a full installation in a Linux environment.

Should a full installation be desirable, then an automated installation script is provided (currently only for RHEL-derived Linux installations) which will attempt to install and configure as much of the software as possible. See section 3.4 (Automated Installation) for details on carrying out a scripted installation.

Experienced Linux users may prefer to use existing installations of prerequisite packages on their systems, or manually install the tools required. Details on manual installation of the prerequisites and configuration of the BugBuilder installation can be found in section 3.5 (Manual Installation).

3.1 Virtual Machine Installation

The BugBuilder virtual machine image has been created with VirtualBox and it is recommended that it be run in a VirtualBox environment, however the disk image can be converted to formats appropriate for running under other virtualised environments such as KVM or VMware.

1. Install VirtualBox from https://www.virtualbox.org, following the instructions for your operating system.
2. Download the BugBuilder virtual machine image from [http://www3.imperial.ac.uk/bioinformatics-support-service/resources/software/bugbuilder](http://www3.imperial.ac.uk/bioinformatics-support-service/resources/software/bugbuilder).

3. Start VirtualBox on your machine - how to do this will vary according to your operating system. Once running you should see a window displayed such as that shown in figure 1.

![VirtualBox main window](image1)

Figure 1: VirtualBox main window

4. Click the 'New' icon on the toolbar to open a dialog box for creating a new virtual machine. Enter 'BugBuilder' in the 'Name' field, then select 'Linux' from the 'Type' drop-down menu, and 'Red Hat (64-bit)' from the 'Version' menu. The window should look like that shown in figure 2. Click the 'Next' button.

![Create virtual machine window](image2)

Figure 2: Create virtual machine window

5. The next window displayed allows the amount of memory available to the virtual machine to be defined. Adjust the slider to set the desired amount of memory as shown in figure 3. BugBuilder has been tested with 4Gb (4096 Mb) RAM allocated to the virtual machine, which appears to be sufficient for most situations. Allocating larger amounts of memory to the virtual machine will result in improved performance, however it is necessary to ensure that the selected value leaves sufficient memory for other programs running on the computer.
3.1 Virtual Machine Installation

Figure 3: Defining memory available to the BugBuilder virtual machine.

6. Next, the hard-disk to be used by the virtual machine needs to be configured. Select the 'Use an existing virtual hard disk file' option, then click the file icon alongside the dropdown menu and navigate to the BugBuilder.vdi file you downloaded in step 2. The window should look like that shown in figure 4. Now click the 'Create' button.

Figure 4: Selecting the virtual disk image.

7. **Network Configuration**: VirtualBox supports a number of different network modes. The network mode can be chosen by selecting the BugBuilder virtual machine in the VirtualBox window, clicking the 'Settings' button on the toolbar then 'Network' in the left-hand pane of the window. The networking mode can then be selected in the 'Attached to:' dropdown menu. (see figure 4).
The default NAT (Network Address Translation) mode will allow the BugBuilder virtual machine to connect to other machines on the internet i.e. to download data onto the virtual machine, however it does not allow you to login to the virtual machine remotely (i.e. using SSH). If you require full network access for the virtual machine, then it will be necessary to use one of the alternate network modes. If your network provides a DHCP server, then 'bridged networking' (which allows the virtual machine access to the network just like a physical machine) is probably the best configuration to use, although this depends upon your network configuration. Please contact your local network administrator to determine the most appropriate way to configure networking on your virtual machine.

### 3.1.1 Starting the Virtual Machine

The virtual machine can be started by selecting the BugBuilder virtual machine in the VirtualBox interface, and clicking the 'Start' icon on the toolbar. The virtual machine will then be booted in a new indow, and when complete will appear as shown in figure 6.

The login screen displays a file-system usage indicator, which shows the amount of space consumed on the virtual machine's disk. By default the disk is configured to expand up to 50Gb in size.

Below this is shown the IP address allocated to the virtual machine. If your networking is configured in a mode which supports connecting directly to the virtual machine via the network (i.e. bridged networking), you can use this IP address to establish your connections.
3.2 Virtual Machine Post-Installation Configuration

The virtual machine is setup to provide a fully working BugBuilder installation. The BugBuilder software is installed in /opt/BugBuilder, with the BugBuilder scripts available on the default path.

Certain packages cannot be pre-installed in the virtual machine image due to licensing constraints, however BugBuilder is perfectly functional without these. The packages which cannot be distributed directly are AbySS, SSpace, GapFiller and RNAmmer. The additional packages can be downloaded from the sites indicated in Table 2, then copied to the /opt/BugBuilder/src directory. The packages can then be installed by running the /opt/BugBuilder/bin/configure.pl command, which will carry out the necessary installation processes and generate an updated configuration file.

3.3 Full Installation

3.3.1 Obtaining the software

The software can be obtained from https://github.com/jamesabbott/BugBuilder as a zip file by clicking the 'Download ZIP' button on the right of the page. Alternatively, if git is installed on your machine you can run git clone https://github.com/jamesabbott/BugBuilder and the latest version of the software will be downloaded into a 'BugBuilder' directory within your current working directory.

3.4 Automated Installation

BugBuilder includes a script to automate the installation of the prerequisite packages within the BugBuilder installation and setup the configuration file appropriately. This has been developed and tested in a CentOS 7 environment but should work on other RHEL-derived distributions. If you want to use existing packages installed on your system, a manual installation and configuration should be carried out.
The location of the extracted software is referred to in the remainder of this document as
$BUGBUILDER_HOME. This can be set as an environmental variable to allow commands in the manual to be
copied verbatim by typing:

export BUGBUILDER_HOME='path'

where 'path' is replaced by the path to the directory where the software is unpacked.

An automated installation can be carried out as follows:

1. Download the BugBuilder software:
   
git clone https://github.com/jamesabbott/BugBuilder

2. Change to the BugBuilder src directory:
   
cd $BUGBUILDER_HOME/BugBuilder/src

3. Download the prerequisites tarball:
   
wget http://web.bioinformatics.ic.ac.uk/BugBuilder/BugBuilder_prerequisites.tar

4. Unpack the downloaded tarball
   
tar xvf BugBuilder_prerequisites.tar

   This will unpack the various software installation packages into the BugBuilder/src directory

5. Remove the tarball to save disk space (optional):
   
rm BugBuilder_prerequisites.tar

6. (Optional) Download additional not-redistributable packages. RNAmmer is an essential prerequisite for
   Prokka, so needs to be installed. The other non-redistributable packages are optional, however it would
   be highly recommended to add these additional packages which can significantly improve final assembly
   contiguity. Additional downloaded packages should be copied into the BugBuilder/src directory. See table 2
   for download details of these packages.

7. Run the automated installation script:
   
$BUGBUILDER_HOME/bin/configure.pl

   The script will firstly attempt to install necessary Perl modules locally within the BugBuilder installation,
   following which the prerequisite software packages will be installed. Each will be unpacked in the src
   subdirectory, then compiled and installed as necessary. Full outputs of the installation process for each
   package being are recorded in the install_logs directory. In the event of any issues with the installa-
   tion a package, a warning will be generated by the configure.pl script, which should be examined to
   determine the cause of the problem.

   Once package installation has completed, an appropriate configuration file will be generated. The loca-
   tion of a directory to use as temporary working space will first be requested (defaulting to tmp). Should
   it not be possible for the script to determine the location of any packages, a prompt will be displayed re-
   questing the path to the installation to be entered. If the package in question is not available, just press
   enter to skip configuration of this package. Following package configuration, the BugBuilder.yaml
   configuration file will be written to the etc directory within the BugBuilder installation, and can then be
   edited using a text editor if required. Details on editing the configuration are contained in the following
   section.
3.5 Manual Installation

3.5.1 Operating System Packages

Certain packages are required which can be installed from the operating systems package repositories. Note that package names may vary between distributions. The necessary packages are indicated in table 1 and can be installed using the appropriate tools according to your operating system (i.e. yum, dnf, apt).

<table>
<thead>
<tr>
<th>Package</th>
<th>RHEL/CentOS/Fedora Package</th>
<th>Debian/Ubuntu Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2</td>
<td>bzip2</td>
<td>bzip2</td>
</tr>
<tr>
<td>gcc</td>
<td>gcc</td>
<td>gcc</td>
</tr>
<tr>
<td>g++</td>
<td>gcc-c++</td>
<td>g++</td>
</tr>
<tr>
<td>fortran</td>
<td>gcc-gfortran</td>
<td>gfortran</td>
</tr>
<tr>
<td>binutils</td>
<td>binutils</td>
<td>binutils</td>
</tr>
<tr>
<td>patch</td>
<td>patch</td>
<td>patch</td>
</tr>
<tr>
<td>libgomp</td>
<td>libgomp</td>
<td>libgomp1</td>
</tr>
<tr>
<td>glibc</td>
<td>glibc.i686, glibc-headers, glibc-devel</td>
<td>libgc6-i386, libgc6-dev</td>
</tr>
<tr>
<td>Perl local::lib</td>
<td>perl-local-lib</td>
<td>liblocal-lib-perl</td>
</tr>
<tr>
<td>CPAN</td>
<td>perl-CPAN</td>
<td>perl-modules</td>
</tr>
<tr>
<td>Perl File::Which</td>
<td>perl-File-Which</td>
<td>libfile-which-perl</td>
</tr>
<tr>
<td>Perl File::Find::Rule</td>
<td>perl-File-Find-Rule</td>
<td>libfile-find-rule-perl</td>
</tr>
<tr>
<td>Perl IO::gzip</td>
<td>perl-PerIO-gzip</td>
<td>libperlio-gzip</td>
</tr>
<tr>
<td>Perl Text::ASCIItable</td>
<td>perl-text-ASCIITable</td>
<td>libtext-asciiitable-perl</td>
</tr>
<tr>
<td>Perl DateTime</td>
<td>perl-DateTime</td>
<td>libdatetime-perl</td>
</tr>
<tr>
<td>Perl Statistics::Basic</td>
<td>perl-Statistics-Basic</td>
<td>libstatistics-basic-perl</td>
</tr>
<tr>
<td>Perl XML::Simple</td>
<td>perl-XML-Simple</td>
<td>libxml-simple-perl</td>
</tr>
<tr>
<td>zlib</td>
<td>zlib, zlib-devel</td>
<td>zlib1g, zlib1g-dev</td>
</tr>
<tr>
<td>Expat</td>
<td>expat, expat-devel</td>
<td>libexpat1, libexpat1-dev</td>
</tr>
<tr>
<td>ntcurses</td>
<td>ntcurses, ntcurses-devel</td>
<td>libncurses5, libncurses5-dev</td>
</tr>
<tr>
<td>readline</td>
<td>readline, readline-devel</td>
<td>libreadline5, libreadline5-dev</td>
</tr>
<tr>
<td>curl</td>
<td>curl</td>
<td>curl</td>
</tr>
<tr>
<td>dos2unix</td>
<td>dos2unix</td>
<td>dos2unix</td>
</tr>
<tr>
<td>sparsehash</td>
<td>sparsehash-devel</td>
<td>libsparsehash-dev</td>
</tr>
<tr>
<td>openMPI</td>
<td>openmpi, openmpi-devel</td>
<td>libopenmpi1.6, libopenmpi1.6-dev</td>
</tr>
<tr>
<td>sqlite3</td>
<td>sqlite, sqlite-devel</td>
<td>sqlite, sqlite-dev</td>
</tr>
</tbody>
</table>

Table 1: Required operating system packages

3.5.2 Prerequisite Software

A number of prerequisite software packages need to be installed prior to running the software, which are summarised in table 2. Where version numbers are specified, these indicate the most recent version of the software which BugBuilder has been tested with. Previous supported versions are indicated in the `etc/package_info.yaml` file. Other versions of these packages will probably work fine, but in case of problems try using the versions listed below. Those which are available under licenses permitting their redistribution (indicated by the "Bundled" column of the table) are available in a tarball. (‘BugBuilder_Prerequisites.tar.gz’) from the BugBuilder website. Non-bundled packages should be downloaded from the locations listed below.
### Table 2: Required software packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Latest Version</th>
<th>Bundled</th>
<th>Required</th>
<th>Download location</th>
</tr>
</thead>
<tbody>
<tr>
<td>fastqc</td>
<td>0.11.4</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://www.bioinformatics.babraham.ac.uk/projects/fastqc">http://www.bioinformatics.babraham.ac.uk/projects/fastqc</a></td>
</tr>
<tr>
<td>sickle</td>
<td>1.210</td>
<td>Y</td>
<td>Y</td>
<td><a href="https://github.com/najoshi/sickle">https://github.com/najoshi/sickle</a></td>
</tr>
<tr>
<td>SPAdes</td>
<td>2.3.0</td>
<td>Y</td>
<td>N</td>
<td><a href="http://bioinf.spbau.ru/spades">http://bioinf.spbau.ru/spades</a></td>
</tr>
<tr>
<td>ABySS</td>
<td>1.9.0</td>
<td>N</td>
<td>N</td>
<td><a href="http://www.bcgsc.ca/platform/bioinfo/software/abyss">http://www.bcgsc.ca/platform/bioinfo/software/abyss</a></td>
</tr>
<tr>
<td>WGS Assembler</td>
<td>8.3rc2</td>
<td>Y</td>
<td>N</td>
<td><a href="http://sourceforge.net/projects/wgs-assembler">http://sourceforge.net/projects/wgs-assembler</a></td>
</tr>
<tr>
<td>AMOS</td>
<td>3.1.0</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://amos.sourceforge.net">http://amos.sourceforge.net</a></td>
</tr>
<tr>
<td>samtools</td>
<td>1.2</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://www.htslb.org">http://www.htslb.org</a></td>
</tr>
<tr>
<td>picard</td>
<td>1.140</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://broadinstitute.github.io">http://broadinstitute.github.io</a></td>
</tr>
<tr>
<td>SIS</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td><a href="http://marte.ic.unicamp.br/8747/">http://marte.ic.unicamp.br/8747/</a></td>
</tr>
<tr>
<td>Mauve</td>
<td>3.0</td>
<td>N</td>
<td>N</td>
<td><a href="http://www.baseclear.com">http://www.baseclear.com</a></td>
</tr>
<tr>
<td>GapFiller</td>
<td>1.10</td>
<td>N</td>
<td>N</td>
<td><a href="http://www.baseclear.com">http://www.baseclear.com</a></td>
</tr>
<tr>
<td>R</td>
<td>3.2.2</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://www.r-project.org">http://www.r-project.org</a></td>
</tr>
<tr>
<td>Prokka</td>
<td>1.5.2</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://www.vicbioinformatics.com/software.prokka.shtml">http://www.vicbioinformatics.com/software.prokka.shtml</a></td>
</tr>
<tr>
<td>Aragorn</td>
<td>1.2.36</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://mbio-serv2.mbiokol.lu.se/ARAGORN">http://mbio-serv2.mbiokol.lu.se/ARAGORN</a></td>
</tr>
<tr>
<td>Prodigal</td>
<td>2.60</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://prodigalorni.gov">http://prodigalorni.gov</a></td>
</tr>
<tr>
<td>HMMER3</td>
<td>3.1b1</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://hmmer.janelia.org">http://hmmer.janelia.org</a></td>
</tr>
<tr>
<td>RNAmer</td>
<td>1.2</td>
<td>N</td>
<td>Y</td>
<td><a href="http://www.cbs.dtu.dk">http://www.cbs.dtu.dk</a></td>
</tr>
<tr>
<td>Infernal</td>
<td>1.1rc2</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://infernal.janelia.org">http://infernal.janelia.org</a></td>
</tr>
<tr>
<td>barmap</td>
<td>0.6.0</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://www.vicbioinformatics.com/software.barmap.shtml">http://www.vicbioinformatics.com/software.barmap.shtml</a></td>
</tr>
<tr>
<td>Blast+</td>
<td>2.2.31+</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://blast.ncbi.nlm.nih.gov/Blast.cgi">http://blast.ncbi.nlm.nih.gov/Blast.cgi</a></td>
</tr>
<tr>
<td>BWA</td>
<td>0.75a</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://bio-bwa.sourceforge.net">http://bio-bwa.sourceforge.net</a></td>
</tr>
<tr>
<td>MUMmer</td>
<td>3.22</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://mummer.sourceforge.net">http://mummer.sourceforge.net</a></td>
</tr>
<tr>
<td>circleator</td>
<td>1.0.0rc4</td>
<td>Y</td>
<td>Y</td>
<td><a href="http://jonathancrabtree.github.io/Circleator/">http://jonathancrabtree.github.io/Circleator/</a></td>
</tr>
</tbody>
</table>

#### 3.5.3 sam2afg

The sam2afg script produced by Shaun Jackman (Canada’s Michael Smith Genome Science Centre) is used in the conversion of assemblies to AMOS’s bank format. This script is available either as part of the ABYSS distribution (where it is named ‘abyss-sam2afg’), or in the AMOS git repository (named ‘sam2afg’), but is not yet available in a released AMOS version. The AMOS tarball available in the prerequisites bundle is therefore based on the git repository which includes this script.

It is necessary to modify this script with the patch located in the ‘src’ directory of the BugBuilder distribution in order for it to be able to use alignments generated using ‘bwa mem’ (which BugBuilder carries out when using reads >100 bp) as follows:

```
[jamesa@codon ~]$ cd /path/to/samtoafg
[jamesa@codon ~]$ patch abyss-samtoafg src/sam2afg.patch
```

This patch has been submitted to the AMOS developers, and hopefully will be made available directly in future AMOS/ABYSS releases which will make this stage unnecessary in future.

#### 3.5.4 Perl Modules

BugBuilder also makes use of some non-core Perl modules which need to be installed. These can be installed either using appropriate operating system packages, where available, or directly from CPAN using CPAN/CPANPLUS modules. The required modules are:

- BioPerl
3.6 Downloading Example Data Sets

A number of example data sets can be downloaded from public repositories to demonstrate the use and capabilities of BugBuilder with readily available data. Run the `bin/download_sample_data` script. When run without arguments, this will display a list of sample data sets which are available to download from the ENA. All available datasets can be downloaded by passing the `-all` argument to the script, otherwise individual datasets can be obtained by passing the `-dataset [name]` argument. By default, example datasets will be downloaded into the `examples` directory within the BugBuilder installation, alternately the destination directory can be specified using the `-dir` argument.

4 Pipeline Workflow

The BugBuilder workflow carries out all the tasks necessary to create an annotated genome assembly from fastq format sequence data. The only other data required is the identity of the sequencing platform used to generate the data, and optionally a reference genome sequence.

4.1 Pipeline Inputs

BugBuilder supports either using reads from fragment libraries i.e. single unpaired reads, or mate-pair libraries. Paired reads should be provided as a pair of non-interleaved fastq files (i.e. in two separate fastq files, rather than one fastq file with alternating read1/read2 sequences). Reads generated with a long-read sequence platform (i.e. PacBio, MinION) can also be provided separately, allowing with stand-alone assembly, or a hybrid assembly with i.e. Illumina reads. Additionally, a fasta format reference sequence from a closely related organism can be provided to assist with scaffolding or orientating the generated contigs, to improve the larger scale structure of the assembly.
4.2 Pipeline Outputs

The pipeline produces annotated contig/scaffold sequences in EMBL format and an AGP 2.0 format file describing the scaffold structure which are appropriate for submission to the ENA. If a reference genome has been provided, comparisons of the assembly against this reference will also be generated using MUMmer-plot and BLAST in a format appropriate for viewing using the Artemis Comparison Tool (www.sanger.ac.uk/resources/software/act/). A graphical map of the completed assembly will also be produced using Circletator.

4.3 Workflow

![Figure 7: BugBuilder Workflow](image)

The overall BugBuilder workflow is shown in Figure 7. The core workflow is indicated by solid lines, with hashed lines indicating optional or conditionally run parts of the pipeline (i.e. depending upon availability of a reference genome sequence).

4.4 Sequence QC

Sequence quality is key to obtaining good de-novo assemblies. A quality assessment of the provided sequence reads will be carried out using FastQC, and the summary results generated displayed on the console. The full FastQC report will be available at the end of the BugBuilder run, and should be examined to determine the cause of any reported QC failures.

Early high-throughput sequence assemblers ignored sequence quality scores, however quality scores are now taken into account by some assemblers. Providing an assembler which does not utilise the quality scores with low quality sequence can have a significantly detrimental effect on the results of the assembly. To ensure that any assemblers which do not use these scores are only given good quality sequence, the sequence reads are first trimmed to a phred-score of Q20, discarded lower quality sequence. Due to the differing error profile inherent in long-read sequence technologies (i.e. PacBio, MinION), sequence trimming is not currently applied to data from these platforms.
4.5 Downsampling

Somewhat counter-intuitively, increasing sequence coverage will not necessarily lead to an improvement in assembly quality. De-bruijn graph based assemblers such as ABySS and SPAdes will typically show improvements in assembly up to around 50x coverage with prokaryotic organisms, with progressively smaller improvements as coverage is increased. Rather than continuing to show improvement at high coverage, assemblies will instead tend to show an increase in incorrect bases, which is a result of errors starting to accumulate at the same loci. This problem can be avoided by downsampling the input sequences to a defined coverage level. In order to determine coverage, BugBuilder needs an estimation of the size of the target genome, which can be determined either from a provided reference genome sequence, or by being passed the `-genome_size` command line argument. If genome coverage is determined to be above a threshold defined by the `-coverage` command line argument (default value: 100), then the reads are downsampled to the required coverage level.

4.6 Assembly

The job of glueing together the huge number of sequence reads generated during a sequencing experiment is handled by a tool known as an assembler. There are now a considerable number of assemblers available capable of producing good assemblies from high-throughput sequencing of microbial genomes. The assembly problem is not entirely solved, however, and new and improved assemblers are regularly being released. Rather than base BugBuilder around a specific assembler, it has instead been developed as a framework which various different assemblers can be plugged into, allowing the assembly component to be readily updated as new algorithms are made available.

There are three main classes of assembler in common usage today, the de Bruijn graph based assembler (i.e. SPAdes, ABySS, Velvet), consensus overlap-graph based assemblers (i.e. Mira, Celera assembler) and string-overlap graph assemblers (i.e. SGA). The different types of assembler tend to work better with different sequence type i.e. de Bruijn assemblers typically produce the best results from short-read data, whereas longer reads will often benefit from an overlap-graph based approach. BugBuilder is made available with configurations for the SPAdes, ABySS and wgs-assembler (CABOG/PBcR) assemblers, however alternative assemblers may have been configured in your installation.

4.6.1 Assembly with Multiple Assemblers

Assemblers typically break contig sequences where sequences become repetitive and difficult to extend unambiguously. Different algorithms will break contigs in different locations, hence it is sometimes possible to get better coverage of a genome by combining multiple assemblies. BugBuilder supports running two independent assemblers, and merging the results (using minimus from the AMOS package). While this approach can produce better genome coverage, it does introduce an addition means of introducing misassemblies, and can result in less contiguous assemblies, albeit covering a greater proportion of a genome.

Note that this functionality was included in BugBuilder prior to the SPAdes assembler being commonly used. SPAdes carries out multiple assemblies using different k-mers which effectively carries out a similar job to BugBuilder's use of multiple assemblers, and typically achieves similar (or better) quality results. This functionality has been retained in BugBuilder since there may be use cases where it is effective, however it would generally be recommended to use SPAdes for assembly in preference to running multiple assemblers.

4.7 Scaffolding

The primary job of an assembler is to build a series of contiguous sequences from the separate sequence reads, extending these sequences as far as possible until it becomes no longer possible to unambiguously extend the contig sequences. Based upon the location of paired reads on different contigs, even when it is no
longer possible to extend a contig sequence any further, it can sometimes be possible to make an association between contigs and get an idea of the size of the gap between them. This is a process known as scaffolding, which is undertaken by some assembly algorithms, but which may also be carried out separately by standalone scaffolding algorithms.

If scaffolds are produced by the selected assembler, these will be available for downstream processes, however if a standalone scaffolder is selected, the outputs of this will be used in preference to those from the assembler.

Scaffolding is the process through which sequence contigs can be arranged to give an impression of the larger-scale organisation of a genome. Typically this is carried out using evidence of association between contigs obtained from read-pairs located on different contigs (see figure 8). This approach tends to have limited success with bacterial genomes sequenced with a single short-insert library since the library insert size is insufficient to span repetitive regions within the genome. An alternative approach which has had considerable success with microbial genomes is to align the generated contigs against the genome sequence of a closely related organism, and use this evidence to order and orientate the contigs.

Figure 8: Scaffolding contigs with read-pair information

As with assembly algorithms, BugBuilder allows scaffolding algorithms to be preconfigured and selected at runtime rather than tying the pipeline into one algorithm. BugBuilder is preconfigured to work with the SSPACE, SIS and Mauve scaffolders. SSPACE utilises read-pair information to generate scaffolds, and with microbial genomes typically does not outperform the scaffolding algorithms build into assembly tools. SIS is a tool which utilises MUMmer to align the contigs against a reference genome, and utilises this information to determine the order and organisation of contig sequences. Mauve takes a similar approach only using it’s own alignment algorithm.

The scaffolder configuration allows for a set of default arguments to be passed to the scaffolder command line, defined using the ‘defaultargs’ attribute. This can be overridden at runtime by passing arguments with the command-line argument ‘-scaffolderargs’.

The advantage of read-pair based scaffolding algorithms is that they are not biased by external factors, whereas alignment against a reference genome may result in an assembly which is artificially skewed towards the organisation of the reference organism. Reference guided scaffolding typically produces far lower number of scaffolds, however.

4.8 Scaffold Orientation and Origin Identification

Some scaffolders will output scaffolds in the correct orientation, but this behaviour varies between scaffolders, and those which just make use of read-pair information have no way of determining the correct orientation of the various scaffolds. If a reference genome is provided, BugBuilder will then align the scaffolds against this reference to ensure they are correctly orientated.

Contigs produced during assembly will typically span the origin of replication. BugBuilder will attempt to locate the origin within the assembled scaffolds based upon the reference genome alignment, and will split the scaffolds at this point, relocating the upstream region to the other end of the assembly.
4.9 Gap Closure

Once scaffolds have been ordered, it can be possible to further close some gaps between contigs where the assembler was not able to unambiguously extend the contig. BugBuilder uses the GapFiller tool produced by BaseClear for this stage of the workflow. GapFiller iteratively aligns reads from the fastq files against the scaffolds, looking for read-alignments which hang off the end of the contiguous sequences. After a number of cycles, some of these extensions may end up spanning the gap between contigs.

Gap closure can be controlled using the \texttt{--gap-fill} command line option. By default, if GapFiller is installed in your BugBuilder installation then gap filling will be carried out on any assembly with less than 150 contigs if a reference genome sequence is provided. For performance reasons, GapFiller will only be run on assemblies containing more than 150 contigs if the \texttt{--gap-fill} command line argument is provided. To prevent GapFiller from running, add the \texttt{--no-gap-fill} argument to the command line.

4.10 AGP Generation

Scaffold structures for genomes submitted to public databases need to be defined in AGP ('A Golden Path') files. This is a text file which species the order and orientation of the contigs on the scaffolds, the sizes of the gaps between contigs and the kind of evidence used to establish the ordering. BugBuilder processes the scaffolds to extract fresh contig sequences by splitting the scaffolds around 'N' s. The public database require a minimum contig size of 200 bp, and do not allow runs of more than 10 'N' s in contig sequences. Contigs shorter than 200 bp are therefore discarded, and if these are within scaffolds the scaffold gaps are extended to take account of the removed sequence. The type of evidence for the gaps is added according to the value of the 'linkage-evidence' attribute in the scaffolder configuration.

4.11 Annotation

The assembled sequences are annotated using the Prokka package, which utilises a number of tools for de-novo CDS, tRNA and rRNA identification. It uses organism specific databases to aid the annotation and can be passed arguments to indicate the genus and species of the genome being annotated. BugBuilder similarly allows these arguments to be specified on the command-line, using the \texttt{--genus} and \texttt{--species} arguments, and passes these values to Prokka if they are provided. If scaffolds have been generated during the assembly process, then the annotation will be applied to them, otherwise the contig sequences will be used.

\begin{verbatim}
BugBuilder --fastq1 read1.fastq --fastq2 read2.fastq --reference myreference.fasta --genus Streptococcus --species pyogenes
\end{verbatim}

Prokka post-processes the annotations to identify terms which do not conform to the NCBI recommendations, and corrects these.

4.12 Outputs

The resulting output files are copied into a directory within the directory from which BugBuilder was launched. If genus/species/strain names have been provided, the directory will be named \texttt{BugBuilder_genus_species_strain}. If the full contents of the working directory are required, for example for debugging purposes, specifying the \texttt{--keepall} command-line argument will copy the full contents of the directory rather than just the final output files.
4.13 Output Visualisation

BugBuilder will additionally generate a number of graphical visualisations of the resulting assembly to assist in interpretation of the outputs.

4.13.1 MUMmerplot

If a reference genome sequence has been provided, an alignment of the assembled scaffolds against the reference sequence will be carried out using MUMmer's nucmer algorithm, which is then used to generate a graphical plot of similarity between the sequences (somewhat akin to a dotplot - see figure 9a). This will be found as a png format image file located in the output directory named `refname_vs_queryname.png`. The reference genome is represented along the X-axis of the plot, and the assembly on the Y-axis, while regions of similarity between the sequences are drawn in red for sequences aligned in the same orientation, or blue for reversed alignments. This provides a useful overview of the large-scale organisation of an assembly in comparison to the reference sequence, and can help identify regions of misassembly. Additionally, a second plot will be generated displaying the percentage identity of the alignment along the length of the reference sequence (figure 9b). This can help identify the degree of conservation present between the genomes.

![Similarity Plot](a) Percentage Identity Plot (b)

Figure 9: MUMmerplot outputs

4.13.2 ACT Comparison

A comparison is also carried out between the reference genome and the scaffolded assembly using NCBI BLAST, output in a tab-delimited format appropriate for loading into the Artemis Comparison Tool (ACT). This allows a fine-grained comparison of the genomes to be carried out, with regions of similarity and reorganisations clearly indicated in the context of the sequence annotations. ACT can be obtained from [http://www.sanger.ac.uk/resources/software/act](http://www.sanger.ac.uk/resources/software/act).

4.13.3 Circleator Visualization

A graphical visualisation of the assembly and gene locations is produced using Circleator [10]. The output is generated as both SVG and PNG format, named `circleator.svg` and `circleator.png`. This is
a circular representation of the assembly, with the scaffolds laid out sequentially around the circle (indicated by dark-blue bands). Surrounding this is a GC plot, while within the scaffold-ring are two rings indicating the location of genes on the positive and negative strands respectively. tRNA locations are highlighted by labels within the circle. The circleator output can be customised by editing the circleator template ($BUGBUILDER_HOME/etc/circleator.cfg.tmp) in accordance with the Circleator documentation [http://jonathancrabtree.github.io/Circleator/documentation.html]. Note that if the Circleator configuration is edited it should be tested by running Circleator outside of BugBuilder to ensure the template is valid and produces the desired results.

![Figure 10: Circleator Genome Visualisation](image)

**5 Submission vs Draft Modes**

BugBuilder can be run in modes targeted at producing output suitable for submission to public databases, or for carrying out further finishing work. The mode to run in is defined using the `-mode` command line argument, which accepts values of either 'submission' or 'draft'. The default mode is 'submission'.

**5.1 Submission Mode**

Submission mode creates output suitable for submission to the ENA, with contigs <200 bp removed, and no consecutive runs of more than 10 'N's in contig sequences. This is the default mode.
5.2 Draft Mode

Draft mode does not remove short contig sequences from the assembly, retaining the full output of the assem-
blier. In order to highlight potential areas of misassembly, the amosvalidate tool from the AMOS package is run
against the assembly. This makes use of mate-pair information to identify potential misassemblies, so can only
be run when a mate-pair library is available. The sequence reads to be mapped against the assembly since the
majority of assemblers do not track the placement of reads in an assembly. Amosvalidate then analyses the
read-placements and identifies features such as read-pairs which are separated by a distance outlying the nor-
mal distribution of the library or regions of unusually high read coverage (potentially indicative of a collapsed
repeat region). These results are stored in Amos's bank format, which can be viewed using the Hawkeye viewer.
The AMOS bank is not returned in the default set of results, so if this is required, the -keepall command-line
argument also needs to be specified. The results are also parsed and included in the EMBL format report,
allowing them to be viewed in the context of the annotations e.g. using Artemis.

Please note that there are always likely to be false-positives in the issues reported, for example by definition,
a small proportion of mate-pairs are always going to be outside the standard-deviation of the insert-size for
the library. The requirement to map the reads against the assembly is itself potentially a source of error. The
mapped location of a read is not necessarily that which was used in the assembly process. The most frequent
regions containing misassemblies are those around repeats, where the read-mapping process is likely to be
least accurate.

6 Running BugBuilder

A list of the assemblers, scaffolders and sequencing platforms defined within the BugBuilder installation can
be obtained by running

[jamesa@codon ~]$ BugBuilder --help

Welcome to BugBuilder

Available assemblers: abyss, spades, celera, PBcR
Available scaffolders: mauve, SIS, sspace
Configured platforms: 454, MinION, PacBio, hybrid, illumina, iontorrent

The simplest usage of BugBuilder is simply to provide the sequence fastq files, along with the name of the
sequencing platform used to generate the reads, and optionally a reference genome sequence. BugBuilder
will select the most appropriate assembler and scaffold based upon the sequencing platform, length of the
supplied reads and whether a reference genome has been provided or not.

BugBuilder --fastq1 read1.fastq --fastq2 read2.fastq \
--reference myreference.fasta --platform illumina

BugBuilder will first evaluate the provided reads and carry out the sequence QC processes, before reporting
on the number of reads available and details of the assembly process and algorithms to be used.

Should a specific assembler be required, then this can be selected using the -assembler command line
argument:

BugBuilder --fastq1 read1.fastq --fastq2 read2.fastq \
--reference myreference.fasta --platform illumina --assembler abyss
6.1 Setting Assembler Parameters

Assemblers typically accept a number of arguments allowing the assembly process to be controlled i.e. by controlling the kmer size used for constructing a de Bruijn graph, or the number of parallel threads to be used. Any arguments to be passed to the assembler by default are defined using the 'default-args' configuration parameter of the assembler definition in the configuration file (see section 7.3). The parameters to be used will be displayed in the summary information output displayed above, however these can be changed at runtime using the --assembler-args argument. The provided arguments will be passed directly to the assembler by appending them to the command line used when it is run. If multiple assembler arguments are being passed, ensure they are surrounded with quotes.

```
BugBuilder --fastq1 read1.fastq --fastq2 read2.fastq --platform illumina \
   --reference myreference.fasta --assembler abyss --assembler-args 'k=31'
```

6.2 Running Multiple Assemblers

Running multiple assemblers (see section 4.6.1) simply requires the --assembler command-line argument to be specified twice:

```
BugBuilder --fastq1 read1.fastq --fastq2 read2.fastq \ 
   --reference myreference.fasta --assembler abyss --assembler spades
```

If additional arguments are to be passed to assemblers when multiple assemblers are being used, specify the --assembly-args twice, with the arguments passed in the same order as the assemblers as requested i.e. the arguments specified by the first use of the --assembly-args argument will be passed to the first assembler requested.

```
BugBuilder --fastq1 read1.fastq --fastq2 read2.fastq \ 
   --reference myreference.fasta --assembler abyss --assembler-args "k=61" \ 
   --assembler spades --assembler-args "-t 8"
```

⚠️ This feature should be used with caution and the results carefully compared to those obtained with running a single assembler since it adds an additional potential source of misassembly, although has produced good results in improving assembly contiguity in certain circumstances. ⚠️

6.3 Specifying Organism Details

The details of the organism undergoing sequencing can be provided to BugBuilder at runtime, which can then be used for

1. Informative naming of output files/directories
2. Passing to Prokka for refining the automated annotation process. Prokka will use the details of the organism to restrict certain database searches based on the taxonomy.
3. Completing the organism details in the scaffolds/contigs.embl.
Organism information can be provided using the -genus, -species and -strain arguments.

When submitting assemblies to the ENA/NCBI/DDBJ databases, a locustag is required, which is used to prefix each gene identifier to ensure they are unique within the databases. Prokka can format identifiers correctly using a locus tag if one is provided, consequently BugBuilder will accept this via the -locustag argument. Locus tags are allocated by the databases e.g. for ENA when registering a sequencing project via Webin an option is provided for reserving a locus tag.

Similarly, the database submission requires the identifier of the sequencing centre producing the data to be supplied. BugBuilder can insert this into the EMBL format annotation output where required if the sequencing centre identifier is provided on the command line using the -centre argument.

An example of a full command line specifying these values looks like:

```
BugBuilder -fastq1 reads_R1.fastq -fastq2 reads_R2.fastq -reference reference.fasta
-platform illumina -genus Eschiricia -species coli -strain K12 -locustag ECK12
```

### 6.4 Tutorial: Running the Example Assemblies

Example datasets can be downloaded from the ENA by running the download_sample_data script. See Section 3.6 (‘Downloading Example Datasets’) for details on running the download script.

#### 6.4.1 Illumina GAII

The example Illumina GAII dataset consists of a 40bp mate-pair library providing 73X coverage of the Staphylococcus aureus EMRSA-16 genome, along with an alternate S. aureus genome as a reference sequence (CP000253). Short reads such as these require a De Bruijn-based assembler. The default BugBuilder configuration provides two algorithms which are applicable to this kind of data - SPAdes and ABySS. By default, the SPAdes assembler will be used, but ABySS can be requested by specifying the '-assembler abyss' argument. A purely de-novo assembly can be carried out by providing just the fastq sequences, and not the reference sequence. In the absence of a reference sequence, SSPACE will be used for scaffolding. It is also necessary to indicate the sequencing platform used to generate the reads through the '-platform' argument, which BugBuilder will use to help determine the choice of assembly and scaffolding algorithm.

```
BugBuilder --fastq1 $BUGBUILDER_HOME/examples/GAII/SRR453031_1.fastq.gz
--fastq2 $BUGBUILDER_HOME/examples/GAII/SRR453031_2.fastq.gz --platform illumina
```

Note that the fastq files download here are suffixed with '.gz', indicating they have been compressed with the gzip algorithm. BugBuilder can handle either uncompressed or gzip compressed fastq files.

If a reference sequence is provided, then this will be used for scaffolding in place of the paired-read based SSPACE, and should provide a considerably more contiguous assembly.

To carry out scaffolding using the default reference-guided scaffold, simply specify the reference sequence:

```
BugBuilder --fastq1 $BUGBUILDER_HOME/examples/GAII/SRR453031_1.fastq.gz
--fastq2 $BUGBUILDER_HOME/examples/GAII/SRR453031_2.fastq.gz --platform illumina
--reference $BUGBUILDER_HOME/examples/GAII/CP000253.fasta
```

The default reference-based scaffold is SIS, with mauve being available as an option which can be requested through the '-scaffold' argument.
BugBuilder --fastq1 $BUGBUILDER_HOME/examples/GAII/SRR453031_1.fastq.gz \  
--fastq2 $BUGBUILDER_HOME/examples/GAII/SRR453031_2.fastq.gz --platform illumina \ --reference $BUGBUILDER_HOME/examples/GAII/CP000253.fasta --scaffolder mauve

Note that the '-assembler' and '-scaffolder' arguments are case-insensitive.

If you wish to force the use of particular assembly and scaffolding algorithms rather than using the defaults, simply ensure these are requested through the command line. BugBuilder will fail with an error message if the selected algorithms are inappropriate for the provided sequencing platform, or read length or scaffolding algorithm type.

Example statistics from these assemblies are shown in table 3.

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Scaffolder</th>
<th>GapFiller</th>
<th>Assembly Size (bp)</th>
<th>No. Contigs</th>
<th>Contig N50 (bp)</th>
<th>No. Scaffolds</th>
<th>Scaffold N50 (bp)</th>
<th>Predicted CDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABySS</td>
<td>-</td>
<td>N</td>
<td>2892176</td>
<td>157</td>
<td>34991</td>
<td>98</td>
<td>51415</td>
<td>2806</td>
</tr>
<tr>
<td>ABySS</td>
<td>Y</td>
<td></td>
<td>2904176</td>
<td>242</td>
<td>51605</td>
<td>98</td>
<td>51065</td>
<td>2698</td>
</tr>
<tr>
<td>ABySS</td>
<td>sspace</td>
<td>Y</td>
<td>2770465</td>
<td>1665</td>
<td>2445</td>
<td>441</td>
<td>11778</td>
<td>2908</td>
</tr>
<tr>
<td>ABySS</td>
<td>SIS</td>
<td>Y</td>
<td>2849991</td>
<td>202</td>
<td>23910</td>
<td>8</td>
<td>1735670</td>
<td>2671</td>
</tr>
<tr>
<td>ABySS</td>
<td>mauve</td>
<td></td>
<td>2899315</td>
<td>252</td>
<td>20598</td>
<td>1</td>
<td>2994078</td>
<td>2721</td>
</tr>
<tr>
<td>SPAdes</td>
<td>-</td>
<td>N</td>
<td>2847197</td>
<td>155</td>
<td>50549</td>
<td>139</td>
<td>50549</td>
<td>2663</td>
</tr>
<tr>
<td>SPAdes</td>
<td>-</td>
<td>Y</td>
<td>2850871</td>
<td>139</td>
<td>50607</td>
<td>139</td>
<td>50607</td>
<td>2666</td>
</tr>
<tr>
<td>SPAdes</td>
<td>sspace</td>
<td>Y</td>
<td>2845325</td>
<td>125</td>
<td>49972</td>
<td>89</td>
<td>58661</td>
<td>2671</td>
</tr>
<tr>
<td>SPAdes</td>
<td>SIS</td>
<td>Y</td>
<td>2796954</td>
<td>112</td>
<td>51595</td>
<td>13</td>
<td>2231102</td>
<td>2594</td>
</tr>
<tr>
<td>SPAdes</td>
<td>mauve</td>
<td></td>
<td>2804688</td>
<td>142</td>
<td>52322</td>
<td>1</td>
<td>2804688</td>
<td>2674</td>
</tr>
</tbody>
</table>

Table 3: GAII Staphylococcus aureus EMRSA-16 assembly results. For comparison, MRSA252 [ENA:BX571856] is a fully sequenced example of an EMRSA-16 isolate with a genome size of 2,902,619 bp, with 2877 annotated CDSs

6.4.2 Illumina MiSeq

The Illumina MiSeq example data set provides 73x coverage of the S.aureus NCTC 8325 genome in 150bp paired reads, with the CP000253 genome as a reference. The longer reads available from the MiSeq permit some different assemblers to be used. The ABySS assembler is still applicable with reads of this length but there are alternatives which can provide a more contiguous assembly. Firstly, the Celera WGS-Assembler is capable of handling reads of this length, although SPAdes typically produces considerably better assemblies so is used as the default. Choices for scaffolding are the same as for the GAII example above. Example results obtained using the various combinations of configured assemblers and scaffolds are indicated in table 4.

BugBuilder --fastq1 $BUGBUILDER_HOME/examples/MiSEQ/SRR1460677_1.fastq.gz \  
--fastq2 $BUGBUILDER_HOME/examples/MiSEQ/SRR1460677_2.fastq.gz --platform illumina \ --reference $BUGBUILDER_HOME/examples/MiSEQ/CP000253.fasta --scaffolder mauve

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Scaffolder</th>
<th>GapFiller</th>
<th>Assembly Size (bp)</th>
<th>No. Contigs</th>
<th>Contig N50 (bp)</th>
<th>No. Scaffolds</th>
<th>Scaffold N50 (bp)</th>
<th>Predicted CDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABySS</td>
<td>-</td>
<td>N</td>
<td>2762081</td>
<td>218</td>
<td>30013</td>
<td>365</td>
<td>30097</td>
<td>2544</td>
</tr>
<tr>
<td>ABySS</td>
<td>Y</td>
<td></td>
<td>2762081</td>
<td>218</td>
<td>30013</td>
<td>365</td>
<td>30097</td>
<td>2544</td>
</tr>
<tr>
<td>ABySS</td>
<td>sspace</td>
<td>N</td>
<td>2761637</td>
<td>220</td>
<td>29914</td>
<td>220</td>
<td>29914</td>
<td>2544</td>
</tr>
<tr>
<td>ABySS</td>
<td>sspace</td>
<td>Y</td>
<td>2767829</td>
<td>159</td>
<td>34446</td>
<td>5</td>
<td>2756010</td>
<td>2520</td>
</tr>
<tr>
<td>ABySS</td>
<td>mauve</td>
<td></td>
<td>2844681</td>
<td>177</td>
<td>34446</td>
<td>1</td>
<td>2844681</td>
<td>2577</td>
</tr>
<tr>
<td>SPAdes</td>
<td>-</td>
<td>N</td>
<td>2782780</td>
<td>81</td>
<td>165407</td>
<td>81</td>
<td>165407</td>
<td>2560</td>
</tr>
<tr>
<td>SPAdes</td>
<td>-</td>
<td>Y</td>
<td>2782889</td>
<td>82</td>
<td>165407</td>
<td>82</td>
<td>165407</td>
<td>2560</td>
</tr>
<tr>
<td>SPAdes</td>
<td>sspace</td>
<td>n</td>
<td>2783287</td>
<td>81</td>
<td>165407</td>
<td>81</td>
<td>165407</td>
<td>2560</td>
</tr>
<tr>
<td>SPAdes</td>
<td>SIS</td>
<td>Y</td>
<td>2747071</td>
<td>41</td>
<td>165496</td>
<td>2</td>
<td>2743691</td>
<td>2529</td>
</tr>
<tr>
<td>SPAdes</td>
<td>mauve</td>
<td></td>
<td>2806317</td>
<td>82</td>
<td>165755</td>
<td>1</td>
<td>2806317</td>
<td>2568</td>
</tr>
</tbody>
</table>

Table 4: MiSEQ Staphylococcus aureus EMRSA-16 assembly results. For comparison, MRSA252 [ENA:BX571856] is a fully sequenced example of an EMRSA-16 isolate with a genome size of 2,902,619 bp, with 2877 annotated CDSs
6.4.3 454

Sequence from 454 and IonTorrent instruments differs considerably from Illumina sequence, not only in read lengths and library insert sizes, but also in the sequences error profile. Whereas errors in Illumina sequence tend to be substitution errors, 454 has a higher proportion of insertions and deletions, in addition to problems determining the lengths of homopolymer runs (consecutive occurrences of the same base) consequently it is necessary to use assemblers which are designed with this error profile in mind. BugBuilder includes the Celera WGS assembler which is appropriate for 454/IonTorrent sequences. The example dataset consists a 454 FLX Titanium fragment library providing 25x coverage of E. coli BW2952. The assembler is named ‘celera’ in the BugBuilder configuration. The data can be assembled without scaffolding by running:

```
BugBuilder --fastq1 $BUGBUILDER_HOME/examples/454/454FLX.fastq.gz \
--platform 454
```

Alternatively, to use the BW2952 genome for scaffolding with SIS, run

```
BugBuilder --fastq1 $BUGBUILDER_HOME/examples/454/454FLX.fastq.gz \
--platform 454 --reference $BUGBUILDER_HOME/examples/454/CP001396.fasta \
--scaffolder SIS
```

or for mauve

```
BugBuilder --fastq1 $BUGBUILDER_HOME/examples/454/454FLX.fastq.gz --platform 454 \
--reference $BUGBUILDER_HOME/examples/454/CP001396.fasta --scaffolder mauve
```

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Scaffold</th>
<th>GapFiller</th>
<th>Assembly Size (bp)</th>
<th>No. Contigs</th>
<th>Contig N50 (bp)</th>
<th>No. Scaffolds</th>
<th>Scaffold N50 (bp)</th>
<th>Predicted CDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celera WGS Assembler</td>
<td>N</td>
<td></td>
<td>4176022</td>
<td>1041</td>
<td>5239</td>
<td>1041</td>
<td>5239</td>
<td>4175</td>
</tr>
<tr>
<td>Celera WGS Assembler</td>
<td>SIS</td>
<td>Y</td>
<td>4256317</td>
<td>1031</td>
<td>5239</td>
<td>2</td>
<td>2748135</td>
<td>4399</td>
</tr>
<tr>
<td>Celera WGS Assembler</td>
<td>mauve</td>
<td>Y</td>
<td>4256417</td>
<td>1031</td>
<td>5239</td>
<td>1</td>
<td>4256417</td>
<td>4311</td>
</tr>
</tbody>
</table>

Table 5: Eschiricia coli BW4005 assemblies. The reference E.coli BW2952 genome is 4578159 bp in size, containing 4079 genes.

6.4.4 PacBio

Long read technologies such as PacBio and MinION offer great advantages for de-novo assembly, since they can generate reads 10s of Kb long, however they are hampered by a comparatively high error rate. These errors are primarily substitution errors, although they are not dependent upon sequence composition, occurring randomly among the sequence. This means that they can be resolved with a comparatively low coverage of the genome. Assembly requires specialised algorithms, one of which is included with the Celera WGS assembler, named PBcR, and is available in ‘the default BugBuilder configuration. An example PacBio data set consisting of 150x coverage of an E. coli K-12 genome is included with the example data downloads, and can be assembled with the command:

```
BugBuilder --fastq1 $BUGBUILDER_HOME/examples/PacBio/PacBio.fastq.gz \
--platform PacBio
```
6.4.5 Hybrid PacBio/Illumina

A hybrid assembly can make use of the benefits of multiple sequencing technologies i.e. a low coverage of long reads (PacBio/MinION), which provide increased contiguity to an assembly, but have a high error rate, combined with the depth and accuracy of e.g. Illumina MiSEQ reads. BugBuilder supports such assemblies, which requires paired fastq reads to be provided, along with a fastq file of long reads. The assembly platform should be specified as 'hybrid'. The default BugBuilder configuration uses the SPAdes assembler for such hybrid assemblies. An assembly of the example E.coli K-12 PacBio dataset along with the MiSeq E.coli K-12 can be carried out as follows:

```bash
BugBuilder --platform hybrid \
--fastq1 $BUGBUILDER_HOME/examples/MiSeq_E_coli_K12/ERR760547_1.fastq.gz \
--fastq2 $BUGBUILDER_HOME/examples/MiSeq_E_coli_K12/ERR760547_2.fastq.gz \
--longfastq $BUGBUILDER_HOME/examples/PacBio/PacBio.fastq.gz \
--reference $BUGBUILDER_HOME/examples/PacBio/U00096.3.fasta \
```

Note that 'real-world' hybrid assemblies would typically use much lower coverages of long-read sequence data than are used in this example.

7 Advanced Configuration

Although BugBuilder is distributed preconfigured for handling the outputs of various sequencing platforms, it is relatively straightforward to add different assemblers and scaffolders to those available in the default configuration. The configuration is defined in a YAML format file located in `$BUGBUILDER_HOME/BugBuilder/etc/BugBuilder.yaml`. This is a plain-text file which can be edited with any text editor installed on the system i.e. gedit, vim.

If you have carried out an automated installation using the `configure.pl` script, a configuration file will have been generated for you complete with the correct locations for the packages installed by the script. Otherwise, a template configuration file can be found in `$BUGBUILDER_HOME/etc/BugBuilder.yaml.tmpl`, which should be copied to `$BUGBUILDER_HOME/etc/BugBuilder.yaml` and edited to complete the configuration.

7.1 Temporary directory location

The first entry in the configuration file `tmp_dir` defines the location of the temporary directory to use. Each BugBuilder run will create a subdirectory within this directory, which will be automatically removed at the end of a successful run. The specified `tmp_dir` location should therefore contain sufficient free space to store the outputs from the expected number of concurrent BugBuilder jobs to be run. The directory will also need to have read, write and execute permissions for any user or group wishing to use BugBuilder.

7.2 Software Installation Locations

The next section of the file contains a list of definitions of where each prerequisite package in installed. These should be set to the directory specified during the installation process for each prerequisite package (i.e. via a `prefix` argument), rather than a bin subdirectory containing the executable files with the installation tree.
If Perl or Python modules required for the execution of any of the dependencies are installed in locations not searched by Perl or Python by default, the location of these can be included using the `perl_lib_path` and `python_lib_path` attributes.

### 7.3 Assembler Configuration

Any assembler which meets some basic criteria can be integrated into the BugBuilder pipeline. The assemblers need to be able to handle non-interleaved, paired fastq formatted sequence reads or single unpaired fastq files. Contig outputs are expected to be in fasta format. Assemblers which output scaffold sequences are also supported, where the scaffolds need to be output in a fasta format containing one record per scaffold, with a stretch of ‘N’s separating the contigs of the sequence.

Assemblers with differing input files requirements, or providing differently formatted outputs can be accommodated within the BugBuilder framework by writing a wrapper script to convert the inputs BugBuilder is able to provide to the required formats, and post-process the outputs to make the suitable for BugBuilders requirements. See the `$BUGBUILDER_HOME/bin/run_celera` script for an example of such a wrapper.

#### 7.3.1 Adding an Assembler

Integrating an assembler is simply a matter of providing an appropriate configuration within the ‘assemblers’ section of the configuration file. See the included `BugBuilder.yaml.template` for examples of complete configurations. A number of attributes are defined within each assembler configuration allowing, for example, the command to be executed to be specified. The values in these attributes are processed at runtime to replace certain strings contained within ‘_’ characters (i.e. ‘__TMPDIR__’) to allow correct filenames and arguments to be passed to the assembler. If the ‘__TMPDIR__’ string occurs in the assemblers attributes, it is replaced at runtime with the path to the temporary directory in use by the BugBuilder run. A list of the supported configuration attributes are defined in table 7 below, while valid runtime template strings are shown in section 7.3.2.

The following attributes may be defined for each assembler:

N.B. Although both the ‘command_se’ and ‘command_pe’ attributes are shown as optional, it is necessary that at least one of these is specified.

#### 7.3.2 Runtime Template Replacements

The following template strings embedded in configuration parameters will be replaced at runtime as follows:

- __TMPDIR__: The fully-qualified path to the working directory
- __FASTQ1__: The name of the ‘read1’ fastq file specified by the `fastq1` command line argument
- __FASTQ2__: The name of the ‘read2’ fastq file specified by the `fastq2` command line argument
- __REFERENCE__: The name of the reference genome fasta file specified by the `-reference` command line argument

### 7.4 Scaffolder Configuration

Adding a scaffolder to the BugBuilder configuration is no different to adding an assembler, and just as with assemblers, there are restrictions on the input and output formats which are required. Scaffolders tend to be slightly more variable than assemblers in how inputs and outputs are specified, so it is more likely that a
### 7.4 Scaffolder Configuration

#### 7.4.1 Defining a Scaffolder

Scaffolders are defined within the 'scaffolders' section of the configuration file. The attributes which may be defined for each scaffolder are listed in Table 8.

#### 7.4.2 Runtime Template Replacements

The following template strings embedded in scaffolder configuration parameters will be replaced at runtime as follows:

- `__TMPDIR__`: The fully-qualified path to the working directory
- `__FASTQ1__`: The name of the 'read1' fastq file specified by the `fastq1` command line argument
- `__FASTQ2__`: The name of the 'read2' fastq file specified by the `fastq2` command line argument
- `__REFERENCE__`: The name of the reference genome fasta file specified by the `-reference` command line argument
- `__INSSIZE__`: The insert size of the library
- `__INSSD__`: The standard deviation of the insert size of the library

---

### Table 7: Assembler configuration arguments

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Y</td>
<td>The name this assembler is referred to as by BugBuilder. This is the value which needs to be specified by the <code>assembler</code> command line argument to select this assembler. It is also used as the name of the subdirectory within the working directory for storing the assembler outputs.</td>
</tr>
<tr>
<td>min_length</td>
<td>Y</td>
<td>Minimum length of read this assembler can process</td>
</tr>
<tr>
<td>max_length</td>
<td>Y</td>
<td>Maximum length of read this assembler can process</td>
</tr>
<tr>
<td>command_se</td>
<td>N</td>
<td>The fully qualified command required to run the assembler with unpaired sequence reads (if supported).</td>
</tr>
<tr>
<td>command_pe</td>
<td>N</td>
<td>The fully qualified command required to run the assembler with paired sequence reads (if supported).</td>
</tr>
<tr>
<td>defaultargs</td>
<td>Y</td>
<td>Additional arguments to pass to the assembler be default. Run-time selection of assembler arguments can be made using the <code>--assemblerargs</code> command-line argument, which overrides the value of this attribute.</td>
</tr>
<tr>
<td>contig_output</td>
<td>Y</td>
<td>The fully qualified name of the fasta file containing contig sequences generated by the assembler.</td>
</tr>
<tr>
<td>scaffold_output</td>
<td>N</td>
<td>The fully qualified name of the fasta file containing scaffold sequences generated by the assembler.</td>
</tr>
<tr>
<td>create_dir</td>
<td>Y</td>
<td>A flag to indicate whether BugBuilder should create a subdirectory for the assembler outputs. Set this to '1' if you need BugBuilder to create a directory (using the assembler name as the name for the directory), or '0' if the assembler creates an output directory itself. If the assembler creates this, the 'command' argument should contain the necessary arguments to set this to the same name as the assembler (see the 'SPAdes' example in the provided configuration). Assemblers which do not support setting the name of the output directory may need to be run via a wrapper script.</td>
</tr>
</tbody>
</table>

wrapper script will be required for scaffolders than assemblers. All the scaffolders configured in the default setup require wrapper scripts (see the `BugBuilder/bin` directory for examples of how these work).
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Y</td>
<td>The name this scaffold is referred to as by BugBuilder. This is the value which needs to be specified by the scaffold command line argument to select this scaffold. It is also used as the name of the subdirectory within the working directory for storing the scaffold outputs.</td>
</tr>
<tr>
<td>command</td>
<td>Y</td>
<td>The fully qualified command required to run the assembler.</td>
</tr>
<tr>
<td>scaffold_output</td>
<td>Y</td>
<td>the fully qualified name of the fasta file containing scaffold sequences generated by the assembler.</td>
</tr>
<tr>
<td>unscaffolded_output</td>
<td>N</td>
<td>the name of the file contig sequences which are not included in scaffolds are written to.</td>
</tr>
<tr>
<td>create_dir</td>
<td>Y</td>
<td>A flag to indicate whether BugBuilder should create a subdirectory for the scaffold outputs. Set this to '1' if you need BugBuilder to create a directory (using the scaffold name as the name for the directory), or '0' if the scaffold creates an output directory itself.</td>
</tr>
<tr>
<td>linkage_evidence</td>
<td>Y</td>
<td>The string to include in the AGP file for linkage evidence between contigs derived from this scaffold. Valid entries are 'paired-ends' (for scaffolders using mate-pair evidence to associate contigs) or 'align-genus' for alignment based approaches (assuming the reference organism is of the same genus). The default value inserted in AGP files is 'paired-ends', since this is the method typically employed by assemblers which have scaffolding capabilities. This attribute is also used to validate the choice of scaffold, such that a 'paired-ends' scaffold requires paired fastq files to be provided, while a scaffold with linkage_evidence of 'align_genus' requires a reference genome to be provided.</td>
</tr>
</tbody>
</table>

Table 8: Scaffold configuration arguments

7.5 Platforms and Assembler Categories

Once the assemblers and scaffolders have been defined, it just remains to create assembler category definitions, which include definitions of the sequencing platforms which can be selected using the `platform` command-line argument. For example, the configuration includes a 'short_illumina' category, which is intended for use with short sequences from older Illumina instruments i.e. GAII, which is defined as follows:

```yaml
assembler_categories:
  - name: 'short_illumina'
    min_length: 25
    max_length: 100
    fixed_length: 1
    platforms:
      - 'illumina'
    assemblers:
      - spades
      - abyss
    scaffolders:
      - SIS
      - sspace
```

The length of reads for which the assembler category is valid is defined by the `min_length` and `max_length` attributes. Appropriate assemblers for use with this sequence type are then assigned via the `assemblers` and `scaffolders` attributes, while finally a platform name is defined ('illumina'), which is the name of the platform which can be specified with the `platform` command-line argument. Note that multiple
assembler categories can include the same platform name, and the most appropriate category will be selected by BugBuilder based upon the length of the provided sequence reads.

7.5.1 Testing the configuration

Following any changes to the configuration file, the `check_config.pl` script should be run, which will check that the necessary software packages are installed and configured correctly, and that the installed versions of the software are supported. Any errors reported by this script should be investigated to determine whether they affect required or optional packages.

8 Command Reference

The following command reference is generated from the embedded documentation in BugBuilder.
8.1 NAME

BugBuilder

8.2 DESCRIPTION

Automated pipeline for assembly of draft quality bacterial genomes with reference guided scaffolding and annotation.

Please see accompanying userguide for full documentation

8.3 REQUIRED ARGUMENTS

platform: Sequencing platform used (e.g. illumina, 454, iontorrent)

8.4 OPTIONAL ARGUMENTS

fastq1: Path to first read of paired library, or fragment library (fastq2: Path to second read of paired library
longfastq: Path to fastq file from long-read sequencer
prefix: Prefix to use for output file naming
reference: Path to fasta formatted reference sequence
assembler: Assembler(s) to run - may be specified twice, in which case the two assemblers will be run in parallel and the results merged using minimus. If no assembler is specified, BugBuilder will try to select an appropriate assembler automatically
assembler-args: Any additional arguments to pass to the assembler. Default values are set in the 'default_args' attribute of the configuration file. If running multiple assemblers, assembler_args should be specified twice, once for each assembler, in the same order than the assemblers are specified.
scaffolder: Scaffolder to run
scaffolder-args: Any additional arguments to pass to the scaffolder. Overides the setting of the 'default_args' setting in the scaffolder configuration
insert-size: Size of insert in paired-read library. This will be determined empirically if a reference genome sequence is provided, so only needs specifying when assembling paired-read sequences for which no reference genome is available.
insert-stddev: Standard deviation of insert in paired-read library. This will be determined empirically if a reference genome sequence is provided, so only needs specifying when assembling paired-read sequences for which no reference genome is available.
genome-size: Approximate genome size. Required for PacBio/MinION assemblies using PBcR
species: Specific name of species if known (i.e. pyogenes). Included in resulting EMBL file, and passed to Prokka during annotation stage.
strain: Name of strain used for inclusion in annotation results.

mode: Mode to run in - valid modes are 'submission' (default) or 'draft'

locustag: Locustag argument to pass to Prokka. Used to customise locus_tag in generated EMBL records.

centre: Sequence centre argument to pass to Prokka. Used to customise locus_tag in generated EMBL records.

[no]-fastqc: Determine whether to run fastqc: Default: on

[no]-trim: Determine whether to quality trim reads. Default: on

trim-qv: Quality threshold for trimming reads. Default: 20

trim-length: Min. length of read to retain following trimming. Default: 50 (25 for reads <50bp)

[no]-split-origin: Determine whether to attempt to split assembly around the origin or not. Should the assembly not be scaffolded using a reference sequence, or where the reference sequence is in a significant number of contigs, then starting the sequence at the origin makes little sense. Default: on

[no]-gap-fill: Determine whether to run GapFiller to close scaffold gaps. Assemblies with a large number of scaffold gaps can result in the gap filling stage taking a significant amount of time. Default: on

keepall: Return full working directory with intermediate files, rather than just returning the results (default off)

help: display short help text

man: display full help text

8.5 REPORTING BUGS


All bug reports should include the output of the `check_config.pl` script, which reports on the installed software packages and versions

8.6 AUTHOR - James Abbott

Email j.abbott@imperial.ac.uk