

Mauve Development Overview

The developers guide contains instructions for compiling the command-line alignment software for both Windows and unix-like systems.

The Mauve graphical interface is written in Java and can be compiled on any system with an installed Java Development Kit.

Windows releases are packaged with the Nullsoft Scriptable Installer System to produce an installer binary.

Source code

The Mauve source code is licensed freely under the [GNU General Public License \(GPL\)](#).

Mauve source code snapshots are produced on a nightly basis and can be downloaded here:

<http://gel.ahabs.wisc.edu/mauve/source/snapshots>

We also maintain a [subversion repository](#) of all source code.

API documentation

API documentation is automatically generated nightly from our [subversion repository](#). Thus, the online API documentation reflects the latest snapshot release and may differ slightly from the most recent official source code release. Browse the [online API documentation](#)

Alternatively, the API documentation can be generated by running the command

`doxygen projects/[doxyfile]` from the top-level directory of any source package

Compiling mauveAligner from source

Although Mauve is provided as a pre-compiled binary for Windows, Linux, and Mac OS X, we also make the source code available so that users can modify Mauve and compile it on other platforms. This chapter describes the procedure for compiling the command-line mauveAligner tool from source code on a unix system. Project files for Microsoft Visual studio are included to compile Mauve in a Windows environment. CodeWarrior project files have been obsoleted.

0. Prerequisites

mauveAligner depends on several additional software packages. The latest code in our subversion repository gets packaged automatically every night and is available from <http://gel.ahabs.wisc.edu/mauve/source/snapshots>. The developers make no guarantee that source snapshots will compile or otherwise work.

libGenome <http://gel.ahabs.wisc.edu/mauve/source> libClustalW <http://gel.ahabs.wisc.edu/mauve/source> libMems <http://gel.ahabs.wisc.edu/mauve/source> mauveAligner <http://gel.ahabs.wisc.edu/mauve/source> muscle_aed <http://gel.ahabs.wisc.edu/mauve/source> pkg-config <http://www.freedesktop.org/software/pkgconfig/releases/> boost <http://www.boost.org>

Each of these packages must be downloaded and installed in order to compile mauveAligner successfully. Alternatively, it may be preferable to check the source code out directly from the sourceforge subversion repository, especially if the code will be modified and changes will be committed back to the repository. To do so, execute the following series of commands inside a development directory:

```
svn co https://mauve.svn.sourceforge.net/svnroot/mauve/libGenome/trunk libGenome
svn co https://mauve.svn.sourceforge.net/svnroot/mauve/libClustalW/trunk libClustalW
svn co https://mauve.svn.sourceforge.net/svnroot/mauve/libMems/trunk libMems
svn co https://mauve.svn.sourceforge.net/svnroot/mauve/muscle/trunk muscle
svn co https://mauve.svn.sourceforge.net/svnroot/mauve/sgEvolver/trunk sgEvolver
svn co https://mauve.svn.sourceforge.net/svnroot/mauve/mauveAligner/trunk mauveAligner
```

1. Installing Boost

Boost may be installed as a pre-compiled RPM on Linux systems, or from source on Windows and Mac OS X. A default build of the boost source may be done using the command `bjam --prefix=$HOME install`. Please see the boost installation instructions at <http://www.boost.org> for more details. When building on Mac OS X, it is recommended to build only static libraries. To do so, use the following bjam command line: `bjam "-sTOOLS=darwin" "-sBUILD=release <runtime-link>static <threading>single/multi"`

2. Installing libGenome

Once downloaded and untarred, libGenome can be compiled and installed using the following commands from within the libGenome directory:

```
./configure --prefix=$HOME
make
make install
```

libGenome is installed by default in the `/usr/local` directory. The `--prefix` argument to `./configure` is optional and specifies the location where libGenome gets installed.

3. Installing libClustalW

The installation procedure for libClustalW is similar to that for libGenome. From the software package's directory, execute the following commands:

```
./configure --prefix=$HOME
make
make install
```

Again, the `--prefix` is an optional argument to `./configure` which specifies the location where the library gets installed.

4. Installing libMems

Since libMems depends on the functionality provided by libGenome, libClustalW, and boost these packages must be installed before libMems. Also, libMems requires the pkg-config software to determine the location of libGenome and libClustalW during the installation process. From the libMems directory, execute the following commands:

```
./configure --prefix=$HOME  
make  
make install
```

Also, if libClustalW or libGenome were installed to a non-standard directory, the directory may need to be added to the PKG_CONFIG_PATH environment variable. For example, if libClustalW were installed in the prefix /home/biogenius, the PKG_CONFIG_PATH variable could be set with this command: `export PKG_CONFIG_PATH="$PKG_CONFIG_PATH:/home/biogenius/lib/pkgconfig"` It's a good idea to add PKG_CONFIG_PATH to the .profile or another login script so it doesn't need to be set manually every time the user logs in. As with the other packages, the installation prefix can be specified using the --prefix argument.

It may be necessary to explicitly tell the configure script the name of some boost libraries. To do so, add --with-boost-filesystem=boost_filesystem-gcc --with-boost-program-options=boost_program_options-gcc to the ./configure command line.

5. Compiling mauveAligner

Once libGenome, libClustalW, and libMems are installed, mauveAligner can be compiled. From the mauveAligner directory, issue the following commands:

```
./configure  
make  
make install
```

In addition to mauveAligner, several supporting applications will be compiled. We do not have explicit documentation for them. Many of them are self-explanatory and give their usage instructions when run without arguments.

The statically linked version of mauveAligner is called mauveStatic. It includes the necessary part of the support libraries directly in the program and can be used on other systems without first installing the support libraries. Note that Mac OS X doesn't support static linking and any applications built on OS X will be dynamically linked.

6. Compiling muscle

mauveAligner and progressiveMauve require a custom build of the MUSCLE sequence alignment program. After downloading and untarring the source from the location described above, execute the following command from within the muscle source code directory:

```
make
```

To use the resulting muscle binary, rename it muscle_aed and copy it to the same directory as the mauveAligner and progressiveMauve binaries.

Compiling from a source snapshot

The latest development snapshots do not have a complete build system. In order to compile snapshots the build system must be regenerated using a recent version of the autotools software. To prepare a source snapshot for a build, execute the following command:

```
./autogen.sh
```

The source directory will now be ready for a build with the usual configure, make, make install procedure.

Other notes

Since Apple's migration to Intel CPUs, Mac software should be compiled for both Intel and PowerPC CPUs, at least for the time

being. Apple refers to programs with both PPC and x86 code as "universal". Building a universal library with GNU autotools is nearly impossible, so it's necessary to build x86 and PPC libraries and binaries separately and then join the result with the lipo tool. To build for a PowerPC CPU on an Intel machine, set the following environment variables:

```
export CFLAGS="-O3 -g -isysroot /Developer/SDKs/MacOSX10.3.9.sdk -arch ppc"
export CXXFLAGS="-O3 -g -isysroot /Developer/SDKs/MacOSX10.3.9.sdk -arch ppc"
```

And compile boost with bjam "-sTOOLS=darwin" "-sGXX=g++ -O3 -g -isysroot /Developer/SDKs/MacOSX10.3.9.sdk -arch ppc" "-sGCC=gcc -O3 -g -isysroot /Developer/SDKs/MacOSX10.3.9.sdk -arch ppc" "-sBUILD=release" <linkflags>-Wl,-syslibroot,/Developer/SDKs/MacOSX10.3.9.sdk <runtime-link>static <threading>single/multi" --prefix=\$HOME install The rest of the build procedure remains the same. The resulting libraries and applications will run on any PowerPC mac with an OS as old as 10.3.9. To create a single universal binary for mauveAligner, run something akin to the following command lipo -create mauveAligner-intel mauveAligner-ppc -output mauveAligner where mauveAligner-intel and mauveAligner-ppc are the paths to the intel and ppc mauveAligner binaries, respectively.

A complete example

The following series of commands will build all libraries and source code from the latest source snapshots on an x86 Linux system, installing software to the user's home directory:

```
# create essential directories if they don't exist
mkdir -p ~/bin
mkdir -p ~/lib
export PATH="$HOME/bin:$PATH"
export LD_LIBRARY_PATH="$HOME/lib:$LD_LIBRARY_PATH"

# make a build directory
mkdir build
cd build

# download the source
wget http://pkgconfig.freedesktop.org/releases/pkg-config-0.20.tar.gz
wget http://internap.dl.sourceforge.net/sourceforge/boost/boost_1_33_1.tar.bz2
wget http://internap.dl.sourceforge.net/sourceforge/boost/boost-jam-3.1.11-1-linuxx86.tgz
wget http://gel.ahabs.wisc.edu/mauve/source/snapshots/libgenome-snapshot.tar.gz
wget http://gel.ahabs.wisc.edu/mauve/source/snapshots/libclustalw-snapshot.tar.gz
wget http://gel.ahabs.wisc.edu/mauve/source/snapshots/libmems-snapshot.tar.gz
wget http://gel.ahabs.wisc.edu/mauve/source/snapshots/muscle-snapshot.tar.gz
wget http://gel.ahabs.wisc.edu/mauve/source/snapshots/mauvealigner-snapshot.tar.gz
wget http://gel.ahabs.wisc.edu/mauve/source/snapshots/sgEvolver-snapshot.tar.gz

# build pkg-config
tar xzf pkg-config-0.20.tar.gz
cd pkg-config-0.20
./configure --prefix=$HOME
make install
export PKG_CONFIG_PATH="$HOME/lib/pkgconfig"
cd ..

# build boost
tar xzf boost-jam-3.1.11-1-linuxx86.tgz
cp cp boost-jam-3.1.13-1-linuxx86/bjam ~/bin/
tar xjf boost_1_33_1.tar.bz2
cd boost_1_33_1
bjam --prefix=$HOME install
cd ..

# build libraries
cd libGenome
./autogen.sh
```

```
./configure --prefix=$HOME && make install
cd ..

cd libClustalW
./autogen.sh
./configure --prefix=$HOME && make install
cd ..

cd libMems
./autogen.sh
./configure --prefix=$HOME --with-boost=$HOME && make install
cd ..

cd mauveAligner
./autogen.sh
./configure --prefix=$HOME && make install
cd ..

cd muscle
make
cp muscle $HOME/bin/muscle_aed
cd ..
cd sgEvolver
./autogen.sh
./configure --prefix=$HOME && make install
cd ..
```

If you will be building the source more than once, or especially if you will be editing and recompiling the source, it will be desirable to have PATH, LD_LIBRARY_PATH, and PKG_CONFIG_PATH set automatically on login. Edit one of \$HOME/.profile or \$HOME/.bashrc or \$HOME/.profile.local to set the environment variables.

To build the GUI, execute the following additional commands:

```
wget http://gel.ahabs.wisc.edu/mauve/source/snapshots/mauve-snapshot.tar.gz
tar xzf mauve-snapshot.tar.gz
cp mauveAligner/src/mauveStatic mauve/linux-x86/mauveAligner
cp mauveAligner/src/progressiveMauveStatic mauve/linux-x86/progressiveMauve
cp muscle/muscle mauve/linux-x86/muscle_aedcd mauve
ant dist
```

The Mauve GUI build will reside in mauve/dist/

Developing the Mauve GUI in Eclipse

The Mauve graphical user interface (GUI) is a Java program that provides a front-end for constructing alignments with the command-line mauveAligner program in addition to offering interactive browsing of the alignment results. Mauve currently requires Java versions 1.4 or later.

The Mauve GUI can be developed in nearly any Java development environment, although we use and recommend [Eclipse](#) version 3 or later. Mauve source code is stored in a subversion repository, which can be accessed via authenticated https using any subversion client, but especially the [Subclipse plug-in](#) from within Eclipse. The brave developer may elect to use a text editor and the included build.xml script for [Apache Ant](#).

What follows is a step-by-step recipe for configuring the standard Mauve build environment:

1. Download and run eclipse

Download from the [eclipse web site](#). Eclipse does not come with a windows installer, so simply run the executable after unpacking it from the compressed archive.

2. Install the subclipse plug-in

Tigris.org maintains [detailed instructions](#) on installing subclipse.

3. Check out the Mauve source code

From within eclipse, go to the "Window->Open perspective->Other..." menu item. A dialog box will appear asking for a perspective selection. Select the "SVN Repository Exploring" perspective. The SVN repository perspective will load. Click the "Add SVN repository" button that appears (it has tiny lettering that says SVN and a + symbol). The svn URL for mauve is <https://mauve.svn.sourceforge.net/svnroot/mauve/mauve>. Upon entering this repository URL, a dialog box appears cautioning that a hostname mismatch exists between the certificate and the server. This is normal and the certificate should be permanently accepted. Assuming all has gone well, the mauve repository should now be registered with Subclipse. At this point the development trunk of mauve can be checked out by expanding the repository (click on the plus), and then right-clicking on "trunk" and checking it out into the workspace. Use the "Check out as project into..." menu item to select the destination directory for the source tree.

4. Build Mauve

Switch back to the Java perspective by again going to the "Window->Open perspective->Other..." menu item and selecting "Java (default)" in the dialog box that appears. If a blue Eclipse welcome screen appears, close it to reveal the Mauve source tree. The code should have already compiled and any warnings or errors will appear in the Problems tab near the bottom of the Eclipse workspace.

5. Install platform-specific aligner binaries for debugging

The Mauve GUI depends on a set of C++ programs that implement the alignment algorithms. These platform specific binaries are mauveAligner, progressiveMauve, and muscle_aed. Copies of these binaries compiled for each of Windows, Linux, and Mac OS X reside in the win32, linux-x86, and osx directories of the Mauve GUI source tree. When launched from within eclipse, Mauve needs a copy of the binaries for the appropriate platform in the top-level repository directory. To install the binaries, copy the appropriate set for your platform to the top level directory (one level up).

6. Run Mauve in the debugger

A debug profile must be created in order to run Mauve in the debugger. Select the "Run->Debug..." menu item and create a new Java Application debug configuration. Name it something like "Mauve debug" and set the main class to org.gel.mauve.gui.Mauve. In the arguments tab, set the Java heap size to something larger than the default. For example, the command -Xmx512m would request 512mb of heap space. Mauve is very memory hungry and 384Mb should probably be considered an absolute minimum

heap allocation. 1024Mb is preferred. Optionally, an alignment or a url of an alignment may be specified as a command-line argument. If specified, the alignment will be loaded upon program launch.

Compiling mauveAligner on Windows

This document describes how to configure a Windows computer to build the command-line mauveAligner and related binary programs. Several software development tools must be installed, all of which are available free of charge as of 11/11/2006. Building parallel OpenMP and 64-bit versions of the code requires a non-free version of Visual Studio Professional. There are two primary ways to build the software, either through the Visual Studio GUI or from a command-line script.

Step 1. Install Visual Studio C++

Download and install the free visual studio c++ express edition and platform SDK step by step instructions:

<http://msdn.microsoft.com/vstudio/express/visualc/usingpsdk/>

Installation must be done as a user with admin rights. Viz studio must be run at least once by an admin after installation to configure paths and registry values.

Alternatively, install a commercial licensed copy of visual studio to do 64-bit and OpenMP builds.

Step 2. Install a subversion client

In this example we will use TortoiseSVN:

<http://tortoisesvn.tigris.org/>

TortoiseSVN requires admin privs to install

Using the command-line build script requires command-line subversion (the apache flavor), available from:

<http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=8100>

Step 3. Install the Boost C++ development libraries

The latest version of boost can be installed with the installers from <http://boost-consulting.com> . At present, the latest version is 1.35.0, available here:

http://www.boostpro.com/boost_1_35_0_setup.exe

Alternatively, boost can be built from source. This is necessary for 64-bit windows boost builds if the installer does not include them. The following instructions are for the 1.35.0 release. Download boost_1_35_0.zip and boost-jam (boost-jam-3.1.13-1-ntx86.zip) from

http://sourceforge.net/project/showfiles.php?group_id=7586

Extract boost to a new development folder (C:Developmentboost). Extract the bjam binary from the boost-jam zip file to the newly created C:Developmentboost directory. Make the changes to vsvars32.bat that are given at the bottom of http://boost.org/tools/build/v1/vc-8_0-tools.html. Admin privileges are necessary to edit vsvars32.bat, and using "Run As..." on your favorite text editor can be helpful here. Next, open a command prompt (Start Menu->Run "cmd.exe"), and switch to the boost folder "cd C:Developmentboostboost_1_35_0". Start the boost build with the command:

```
bjam --prefix=C:Developmentboostboost_1_35_0" --toolset=msvc release address-model=32 runtime-link=static link=static  
threading=multi --with-filesystem --with-iostreams --with-program_options --with-system -a install
```

```
bjam --prefix=C:Developmentboostboost_1_35_0" --libdir=C:Developmentboostboost_1_35_0lib64" --toolset=msvc release  
address-model=64 runtime-link=static link=static threading=multi --with-filesystem --with-iostreams --with-program_options  
--with-system -a install
```

The build may take hours to complete on a slow machine. Once completed, the libraries will reside in C:Developmentboost and the build directories can be deleted (boost_1_35_0stage and boost_1_35_0bin) to save disk space.

Step 4 (GUI only). Check out Mauve-related source code repositories

If you are only interested in the automated build script, skip to Step 4 below. Otherwise, create a development folder e.g. "C:Development" and right-click inside the development folder. Select the "SVN checkout" menu item. A dialog box will pop up asking the location of the repository to check out, and the destination directory. Perform a checkout for each of the following

repositories:

Repository Checkout Directory

<https://mauve.svn.sourceforge.net/svnroot/mauve/>

Step 5 (GUI only). Configure Visual Studio paths

In order to build mauve-related software, it is necessary to set the include header and library search paths. To do so, launch visual studio and from the menu bar, select "Tools->Options..." In the dialog that appears, select the "Projects and Solutions"->"VC++ Directories" panel. Then select "Show directories for:"->"Include files" in the preference panel. Now click the new folder icon and add the following directories:

- * C:DevelopmentlibGenome
- * C:DevelopmentlibClustalW
- * C:Developmentmuscle
- * C:DevelopmentmusclelibMUSCLE
- * C:DevelopmentlibMems
- * C:Developmentboostboost_1_35_0

If source code was installed in a directory other than C:Development, change the paths accordingly. Then select directories for library files ("Show directories for:"->"Library files") and add the following paths:

- * C:DevelopmentlibGenomelib
- * C:DevelopmentlibClustalWlib
- * C:Developmentmusclelib
- * C:DevelopmentlibMemslib
- * C:Developmentboostboost_1_35_0lib

On 64-bit windows, replace the boost library path with the path to 64-bit libraries, e.g. "C:Developmentboostboost_1_35_0lib64"

Step 6 (GUI only). Build mauveAligner, progressiveMauve, procrastAligner, muscle etc.

Open the file mauveAligner/projects/everything.sln with visual studio. The workspace contains an assortment of projects related to mauve. To build all projects, right-click on the text "Solution 'everything'" in the project listing at left and select "Build Solution" from the popup menu. A batch build can be used to build for several targets at once (menu "Build"->"Batch Build"). When the build succeeds, binaries will generally be in the mauveAligner/projects or mauveAligner/bin directories.

Step 4 (Script only). Configure and run the build script

Download the latest windows build batch script from:

http://mauve.svn.sourceforge.net/svnroot/mauve/build_scripts/build_win.bat

Save the batch script to a temporary build directory, for example C:build_temp. Then right click on the build script to edit it. You must now configure the program paths near the top of the build script. Once you have finished doing so, simply run the script from a windows command-prompt: build_win.bat

Packaging and deploying Mauve releases

The Mauve source tree includes scripts to support packing Mauve releases on Windows, Linux, and Mac OS X. In general, a computer running each OS is required to package the corresponding release. It is not possible, for example, to cross-package a Mac OS X release from within Linux.

Packaging Linux releases

1. Download the Mauve GUI source code from subversion or the Mauve web site
2. Ensure that the correct pre-compiled aligner binaries exist in the linux-x86 folder
3. Set the release version in the top-level build.xml file
4. Execute the command ``ant dist`` from the top-level mauve source directory
5. A release file called `mauve_linux_${version}.tar.gz` will be created in the dist directory

Packaging Mac OS X releases

1. Download the Mauve GUI source code from subversion or the Mauve web site
2. Ensure that the correct pre-compiled aligner binaries exist in the osx folder
3. Set the release version in the top-level build.xml file
4. Execute the command ``ant macdist`` from the top-level mauve source directory
5. A release file called `Mauve-${version}.dmg` will be created in the dist directory

Packaging Windows releases

Windows release packaging requires the freely available [Nullsoft Scriptable Installation System](#) . Please download and install this software prior to packaging Mauve installers for Windows.

1. Download the Mauve GUI source code from subversion or the Mauve web site
2. Ensure that the correct pre-compiled aligner binaries exist in the win32 folder
3. Set the release version in the top-level build.xml file
4. Download the latest win32 Java online installer from java.com to the top-level mauve directory. Update mauve.nsi to contain the path to latest installer (currently `jre-1_5_0_09-windows-i586-p-iftw.exe`)

5a. From within Eclipse, run the 'nsicompile' target of build.xml

OR

5b. From the windows command-line, execute ``ant nsicompile``

6. A release file called `mauve_installer_${version}.dmg` will be created in the dist directory

Deploying releases

Additional ant targets exist in the build.xml file called `deployLinux`, `deployWin32`, and `deployMacOSX`. These targets assume that your workstation is directly connected to the Genome Evolution Lab network and that ssh private keys have been installed to permit password-free authentication. Unless you are me or you hack our network, you will not be able to automatically deploy releases.

Deploying ASAP-integrated Mauve releases

Mauve includes an applet that allows the Mauve Java Web Start application to communicate directly with the ASAP annotation database. To build an ASAP-integrated Mauve release, it will be necessary to either create or obtain Java code signing certificates. Once a certificate has been obtained, the `key.keystore` variable in `build.xml` should be modified to reflect the location of the keystore file. The ASAP deployment code in `build.xml` assumes that the deployment takes place from a machine running Windows, deployed to a web server running Linux. The variables `asap.dir` and `asap.codebase` in `build.xml` must be configured appropriately. `asap.dir` is the location of the web server directory where the production copy of the Mauve applet and application reside. It must be accessible as a UNC path within windows. This directory can be a symlink on the web server from the user's home directory to the production directory. `asap.codebase` is the base URL where the newly installed Mauve application and applet will reside. Once these variables have been configured correctly and the keystore is in place, the `deployASAP` target of `build.xml` can be executed. Note that the password for the keystore must be provided when the target is executed. This can be done either on the `java -jar ant.jar` command line, or from within eclipse by providing a new VM Argument in the JRE tab of the "Run as...->Ant build..." dialog box. The vm argument to add is `-Dkey.password=<your_keystore_password>`

Evaluating alignment quality and stress-testing the aligner

Purpose

When constructing a sequence alignment system various design decisions can have an impact on sequence alignment quality. To assess the effects each of these decisions can have on alignment quality, we have created a method to score alignment accuracy across numerous test cases using genome sequences that have diverged to varying degrees. We refer to such test results as "accuracy profiles" because they demonstrate the aligner's average accuracy over many evolutionary scenarios.

Overview of the methodology

- * simulate evolution
- * align simulated genomes
- * score the alignment
- * do this many times using different parameter combinations to characterize the aligner's behavior in the presence of different types of evolution
- * Use a high-throughput computing environment such as [Condor](#) to quickly process many simulations

Using the evolver software

A pre-compiled Windows executable is available here: <http://gel.ahabs.wisc.edu/mauve/downloads/sgEvolver.exe>

The following components are necessary to compile the evolver software under Windows, Linux, or Mac OS X:

libGenome <http://www.libgenome.org> libMems <http://gel.ahabs.wisc.edu/mauve/source> sgEvolver
<http://gel.ahabs.wisc.edu/mauve/source> mauveAligner <http://gel.ahabs.wisc.edu/mauve/source> pkg-config
<http://www.freedesktop.org/software/pkgconfig/releases/wxWidgets> <http://www.wxwidgets.org>

The following additional software supports automated aligner testing and generation of postscript accuracy plots:

seq-gen <http://evolve.zoo.ox.ac.uk/software.html?id=seqgen> R <http://r-project.org> perl <http://perl.org>

All of these packages must be installed in order to perform aligner accuracy profiling. Some of these packages may already be installed on your system.

Necessary hardware

Genome alignment software is notoriously CPU intensive. Performing all but the most rudimentary set of alignment experiments will require significant computation resources. This software is designed to work on large compute clusters with the condor job scheduler, however submission scripts for other scheduling systems should be straightforward.

Compiling sgEvolver

- * Follow instructions to [compile mauveAligner from source](#)
- * download sgEvolver source
- * configure --with-libGenome=/prefix/to/libGenome
- * make
- * make install

Compiling from a source snapshot on unix

The nightly subversion source code snapshots do not include a distributable build system. In order to build from the snapshots you will need the GNU autotools (e.g. automake, autoconf, etc.) installed on your system. After unpacking the tarball, execute the following sequence of commands to generate the build system: libtoolize && aclocal && autoheader && automake -a && autoconf. The binaries can then be built using the usual ./configure, make, make install procedure.

How to simulate evolution and test an aligner's accuracy:

- 1) Install the software listed above
- 2) Create a new directory for the simulation experiment and make a copy of `simujobparams.pm` in this directory.
- 3) Create a raw ancestral sequence file. The ancestral sequence must be at least twice as long as the size of genomes to be evolved, preferably longer. It must also be a "raw" sequence file, meaning that it has no Multi-FastA formatting and no deflines: just the

characters { A,C,G,T,a,c,g,t } all on a single line. A utility application called toRawSequence is included with mauveAligner that can convert many major sequence formats to raw sequence.

- 4) Edit simujobparams.pm appropriately. At the very least be sure to set the name of the ancestral sequence file and the locations where software is installed. The simujobparams.pm file has detailed instructions describing each variable.
- 5) Run simujobgen.pl Running the simujobgen script will create a number of alignjob directories, each of which corresponds to a simulation and alignment job with a different combination of mutation parameters.
- 6) change into one of the alignjob directories
- 7) Run simujobrun.pl <aligner> where <aligner> is the aligner to use and can be one of none, mauve, mavid, mlagan, or slagan, assuming these aligners have been installed correctly. When none is selected, the procedure generates the evolved genomes without trying to align and score them.
- 8) In case something goes wrong, check the files command_lines.txt and all the files ending in .err. The .err files correspond to each program's standard error output.
- 9) If none was selected for your aligner, then the evolved sequences will reside in the Multi-FastA file evolved_seqs.fas, and an alignment of orthologous regions will reside in evolved.dat. If an aligner was tested then the alignment scores are reported in the file scores.txt and the scores on regions conserved among all genomes (backbone) are in the file bb_scores.txt

Using the Condor High Throughput Computing environment to process many simulations rapidly

- 1) Follow steps 1 through 5 above. When editing simujobparams.pm be sure to set the paths to the aligners and the scoring tools. These paths must be accessible from the compute node running the job, thus they should be on shared storage. The simujobrun.pl script executes programs in order to carry out simulated evolution, alignment, and scoring of alignments. In particular, evolution requires dd, seq-gen, and sgEvolver. Alignment requires an aligner, and scoring requires scoreAlignment and extractBackbone.
- 2) Step 5 will create a condor DagMan submission script called jobs.dag and a job submission script called mauveAlign.condor. Edit the mauveAlign.condor submission script to select the aligner you would like to test. Options are mauve, mavid, mlagan, slagan, and none. Optionally, the debug parameter may also be given to simujobrun. When debug is used none of the data files generated during the simulated evolution and alignment process are deleted and all get sent back to the job submission host. It may be necessary to set other condor-specific parameters as well.
- 3) Submit the condor jobs with condor_submit_dag -maxjobs ## jobs.dag Here ## is the maximum number of condor jobs that will run simultaneously.
- 4) When all jobs have completed successfully, use scoregen.pl to extract scores from the alignjob directories and generate a heat plot in postscript format. scoregen.pl depends on rgradientplot.R in your tools directory. When running multiple replicates of the simulation, scoregen.pl will automatically average together the scores for each replicate and plot a single average score for that combination of mutation rates.