MAKER Tutorial for WGS Assembly and Annotation Winter School 2018

Contents

About MAKER

MAKER is an easy-to-use genome annotation pipeline designed to be usable by small research groups with little bioinformatics experience. However, MAKER is also designed to be scalable and is thus appropriate for projects of any size including use by large sequencing centers. MAKER can be used for *de novo* annotation of newly sequenced genomes, for updating existing annotations to reflect new evidence, or just to combine annotations, evidence, and quality control statistics for use with other GMOD programs like GBrowse, JBrowse, Chado, and Apollo.

MAKER has been used in many genome annotation projects (these are just a few):

- *Pinus taeda* - Loblolly Pine - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/24653211)
- *Pinus lambertiana* - Sugar Pine - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/27794028)
- *Fusarium circinatum* - Pine Pitch Canker - ResearchGate (https://www.researchgate.net/publication/220022290_First_fungal_genome_sequence_from_Africa_a_preliminary_analysis)
- *Latimeria menadoensis* - African Coelacanth - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/23598338)
- *Atta cephalotes* - Leaf-cutter Ant - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/21347285)
- *Linepithema humile* - Argentine Ant - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/21282631)
- *Pogonomyrmex barbatus* - Red Harvester Ant - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/21282651)
- *Solenopsis invicta* - Fire Ant - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/21282665)
- *Pythium ultimum* oomycete - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/20626842)
- *Petromyzon marinus* - Sea Lamprey annotation and re-annotation - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/23435085)
- *Zea mays* - maize re-annotation - PubMed (https://www.ncbi.nlm.nih.gov/pubmed/25384563)

There are many more projects that use MAKER around the world.

Projects using MAKER around the globe

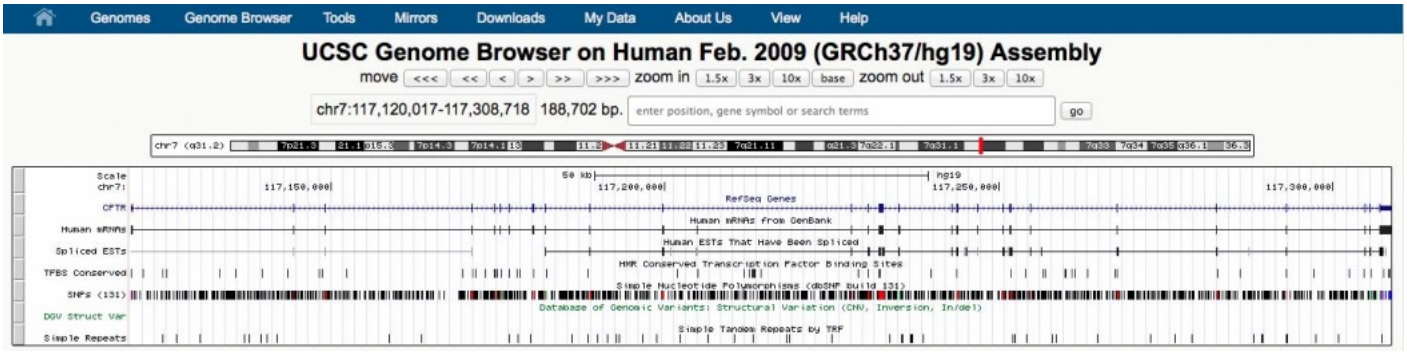
Introduction to Genome Annotation

What Are Annotations?

Annotations are descriptions of different features of the genome, and they can be structural or functional in nature.

Examples:

- Structural Annotations: exons, introns, UTRs, splice forms (Sequence Onotology (http://www.sequenceontology.org/))



Structural Annotations

- Functional Annotations: process a gene is involved in (metabolism), molecular function (hydrolase), location of expression (expressed in the mitochondria), etc. (Gene Ontology (http://www.geneontology.org/))

Select all | Clear all | Perform an action with this page's selected terms... | Go!

Accession, Term		Ontology	Qualifier	Evidence	Reference	Assigned by
GO:0015701 : bicarbonate transport	57 gene products view in tree	biological process		IEA With Ensembl:ENSRNOP00000010981	GO REF:0000019	Ensembl (via UniProtKB)
GO:0032870 : cellular response to hormone stimulus	3283 gene products view in tree	biological process		IEA With Ensembl:ENSRNOP00000010981	GO REF:0000019	Ensembl (via UniProtKB)
GO:0006695 : cholesterol biosynthetic process	263 gene products view in tree	biological process		IEA With Ensembl:ENSMUSP00000049228	GO REF:0000019	Ensembl (via UniProtKB)
GO:0030301 : cholesterol transport	406 gene products view in tree	biological process		IEA With Ensembl:ENSMUSP00000049228	GO REF:0000019	Ensembl (via UniProtKB)
GO:0015705 : iodide transport	17 gene products view in tree	biological process		IEA With Ensembl:ENSRNOP00000010981	GO REF:0000019	Ensembl (via UniProtKB)
GO:0030324 : lung development	1244 gene products view in tree	biological process		IEA With Ensembl:ENSRNOP00000010981	GO REF:0000019	Ensembl (via UniProtKB)
GO:0045909 : positive regulation of vasodilation	97 gene products view in tree	biological process		IEA With Ensembl:ENSRNOP00000010981	GO REF:0000019	Ensembl (via UniProtKB)

Functional Annotations

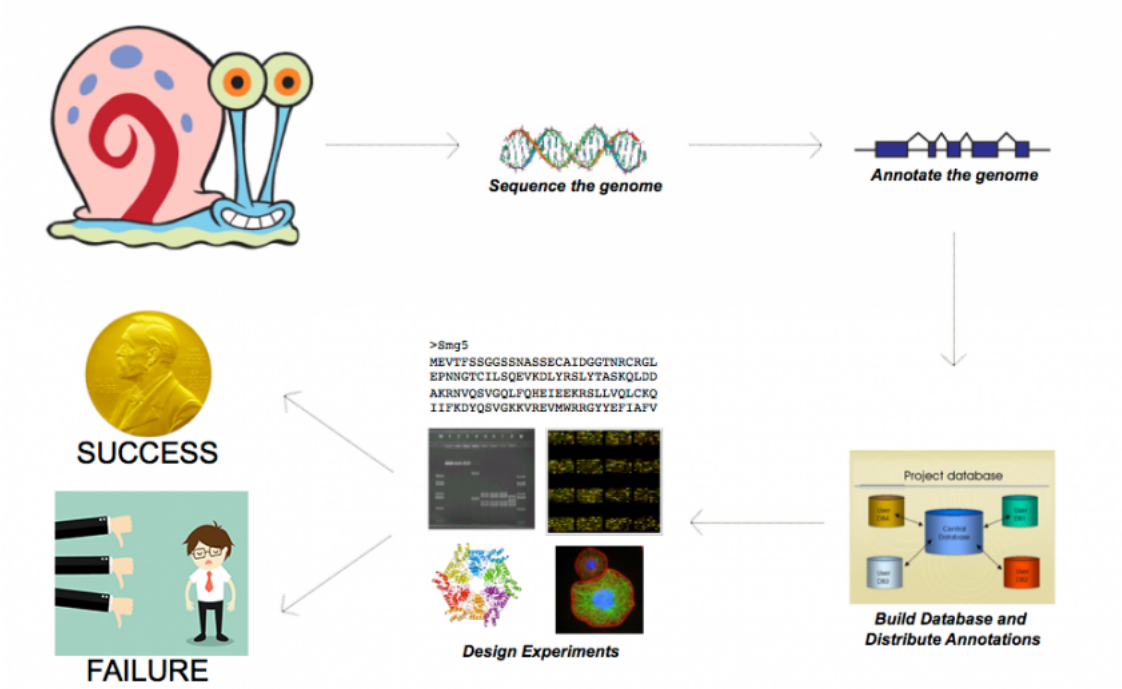
It is especially important that all genome annotations include an evidence trail that describes in detail the evidence that was used to both suggest and support each annotation. This assists in curation, quality control and management of genome annotations.

Examples of evidence supporting a structural annotation:

- *Ab initio* gene predictions
- Transcribed RNA (mRNA-Seq/ESTs/cDNA/transcript)
- Proteins

Importance of Genome Annotations

Why should the average biologist care about genome annotations?



Genome project from sequencing to experimental application of annotations

Genome sequence itself is not very useful. The first question that occurs to most of us when a genome is sequenced is, "where are the genes?" To identify the genes we need to annotate the genome. And while most researchers probably don't give annotations a lot of thought, they use them everyday.

Examples of Annotation Databases:

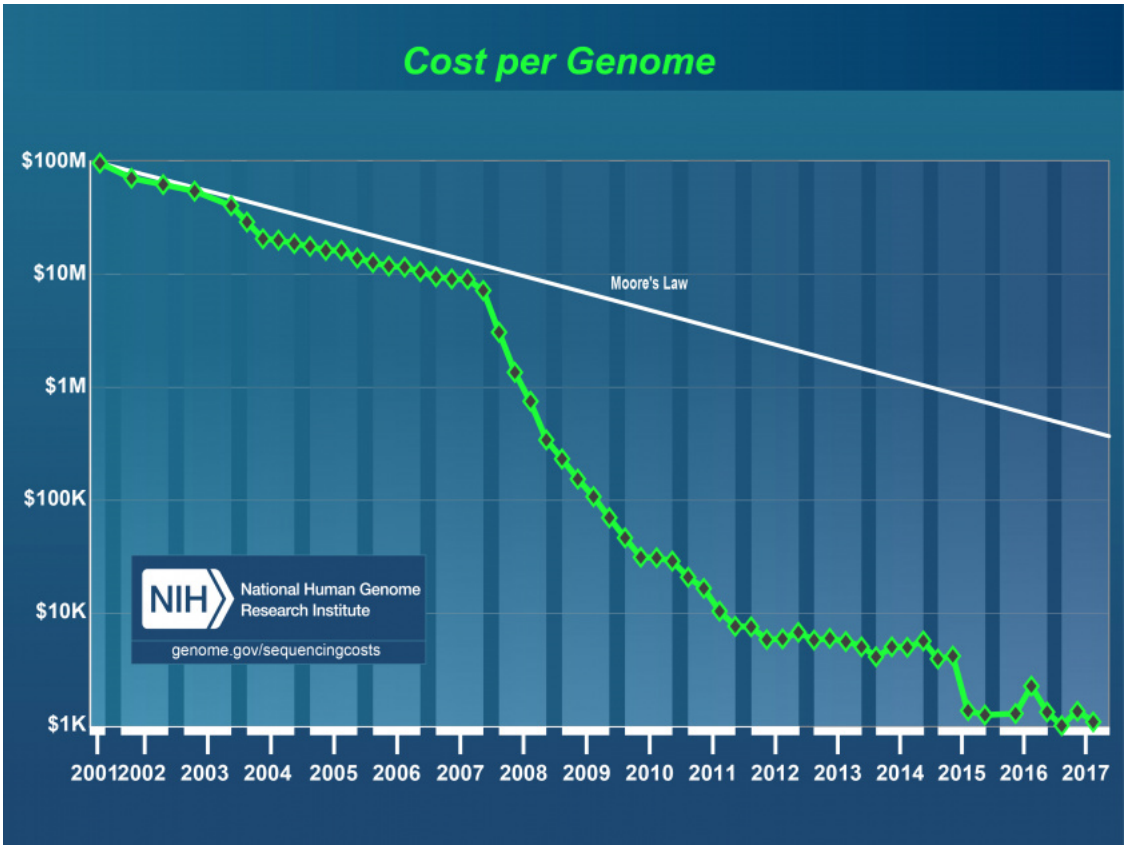
- Ensembl (http://uswest.ensembl.org/index.html)
- RefSeq (http://www.ncbi.nlm.nih.gov/RefSeq/)
- FlyBase (http://flybase.org/)
- WormBase (http://www.wormbase.org/)
- Mouse Genome Informatics (http://www.informatics.jax.org/)

Every time we use techniques such as RNAi, PCR, gene expression arrays, targeted gene knockout, or ChIP we are basing our experiments on the information derived from a digitally stored genome annotation. If an annotation is correct, then these experiments should succeed; however, if an annotation is incorrect then the experiments that are based on that annotation are bound to fail. Which brings up a major point:

- **Incorrect and incomplete genome annotations poison every experiment that uses them.**

Quality control and evidence management are therefore essential components to the annotation process.

Effect of NextGen Sequencing on the Annotation Process



It is now possible to sequence even human sized genomes for as little as $1,000. As costs have dropped, read lengths have increased, and assembly and alignment algorithms have matured, the genome project paradigm is shifting. Even small research groups are turning their focus from the individual reference genome to the population. This shift in focus has already lead to great insights into the genomic effects of domestication and is very promising in helping us understand multiple host-pathogen relationships. Importantly, these population-based studies still require a well-annotated reference genome. Unfortunately, advances in annotation technology have not kept pace with genome sequencing, and annotation is rapidly becoming a major bottleneck affecting modern genomics research.

For example:

- As of January 2018, 8,955 Eukaryotic genome projects were at various stages of completion (4,683 were still being sequenced and 4,272 had at least a draft assembly, but not necessarily gene annotations).
- If we assume just 10,000 genes per genome, that's almost 90,000,000 new annotations (with this many new annotations, quality control and maintenance is also a major issue).
- There are an additional 82,859 prokaryotic genome projects with various stages of completion with hundred of millions of additional potential gene annotations.
- While there are organizations dedicated to producing and distributing genome annotations (i.e ENSEMBL, JGI, Broad), the shear volume of newly sequenced genomes exceeds both their capacity and stated purview.
- Small research groups are affected disproportionately by the difficulties related to genome annotation, primarily because they often lack bioinformatics resources and must confront the difficulties associated with genome annotation on their own.

MAKER is an easy-to-use annotation pipeline designed to help smaller research groups convert the current tsunami of genomic data provided by next generation sequencing technologies into a usable resource.

MAKER Overview

MAKER

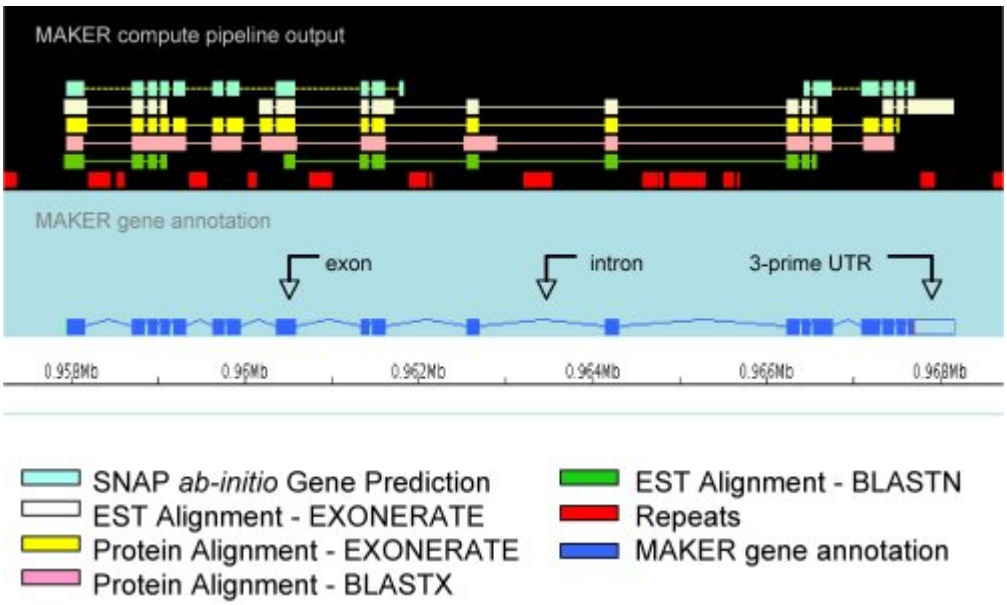
The easy-to-use annotation pipeline.

Annotate this!

User Requirements:	Can be run by small groups (single individual) with a little linux experience
System Requirements:	Can run on desktop computers running Linux or Mac OS X (but also scales to large clusters)
Program Output:	Output is compatible with popular GMOD annotation tools like Apollo, GBrowse JBrowse
Availability:	Free, open-source application (academic use)

What does MAKER do?

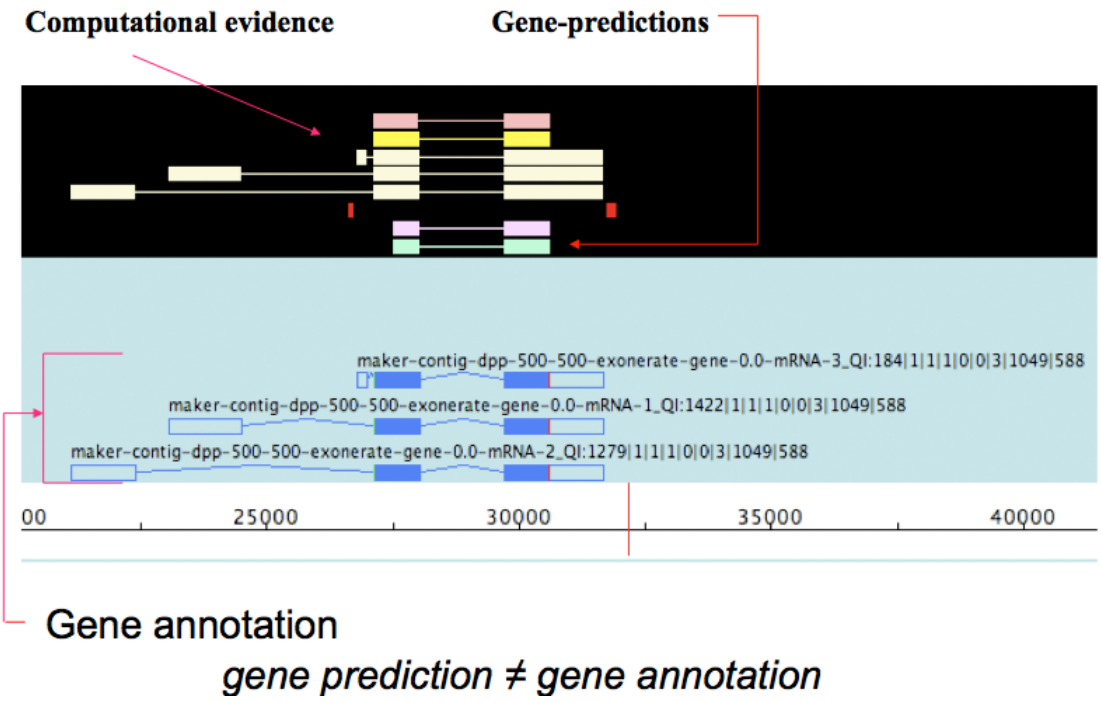
- Identifies and masks out repeat elements
- Aligns ESTs to the genome
- Aligns proteins to the genome
- Produces *ab initio* gene predictions
- Synthesizes these data into final annotations
- Produces evidence-based quality values for downstream annotation management



MAKER-generated annotations, shown in Apollo

What sets MAKER apart from other tools (*ab initio* gene predictors etc.)?

MAKER is an annotation pipeline, not a gene predictor. MAKER does not predict genes, rather MAKER leverages existing software tools (some of which are gene predictors) and integrates their output to produce what MAKER finds to be the best possible gene model for a given location based on evidence alignments.



gene prediction ≠ gene annotation

- gene predictions are gene models.
- gene annotations are gene models but should include a documented evidence trail supporting the model in addition to quality control metrics.

This may seem like a matter of semantics since the output for both *ab initio* gene predictors and the MAKER pipeline are conceptually the same - a collection of gene models. However there are significant differences that are discussed below.

Emerging vs. Classic Model Genomes

Not all genomes are created equal - each comes with its own set of issues that are not necessarily found in classic model organism genomes. These include difficulties associated with repeat identification, gene finder training, and other complex analyses. Emerging model organisms are often studied by small research communities which may lack the infrastructure and bioinformatics expertise necessary to 'roll-ther-own' annotation solution.

'Old School' Model Organism Annotation	'New' Emerging Model Organism Annotation
Well developed experimental systems	The genome will be a central resource for experimental design
Much prior knowledge about genome/transcriptome/proteome	Limited prior knowledge about genome
Large community	Small communities
$$$	$
Examples: *D. melanogaster*, *C. elegans*, human	Examples: oomycetes, flat worms, cone snail

Comparison of Algorithm Performance on Model vs. Emerging Genomes

If you have looked at a comparison of gene predictor performance on classic model organisms such as *C. elegans* you might conclude that *ab initio* gene predictors match or even outperform state of the art annotation pipelines, and the truth is that, with enough training data, they do very well. It is important to keep in mind, however, that *ab initio* gene predictors have been specifically optimized to perform well on model organisms such as *Drosophila* and *C. elegans*, organisms for which we have large amount of pre-existing data to both train and tweak the prediction parameters.

Gene model accuracy for gene prediction/annotation programs

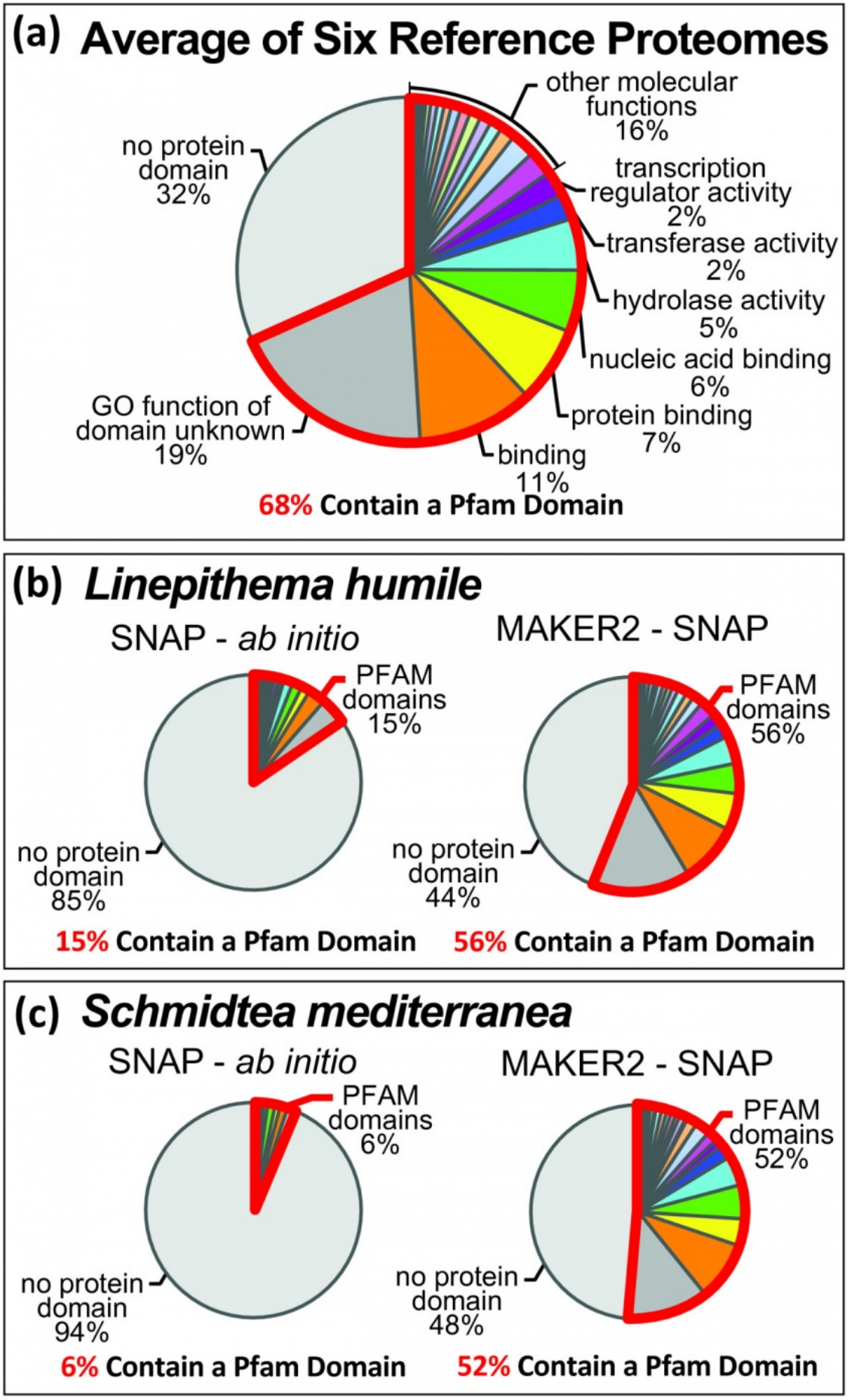
Reference Organism	Performance Category	*Ab Initio* Predictions			MAKER Annotations		
		Augustus	GeneMark	SNAP	Augustus	GeneMark	SNAP
A. thaliana	Nucleotide Accuracy	77.04%	74.68%	69.78%	80.53%	79.39%	80.27%
	Exon Accuracy	67.03%	61.31%	56.40%	67.81%	69.60%	68.78%
D. melanogaster	Nucleotide Accuracy	76.08%	66.54%	69.29%	76.42%	73.66%	74.33%
	Exon Accuracy	61.37%	47.31%	47.01%	58.56%	58.03%	58.49%
C. elegans	Nucleotide Accuracy	88.29%	88.09%	85.10%	87.14%	86.29%	88.48%
	Exon Accuracy	74.62%	68.88%	61.38%	68.60%	65.03%	66.19%

Comparison of gene model accuracies in first-generation genomes for the *ab initio* gene predictors Augustus, GeneMark, and SNAP in comparison to gene model accuracies produced by the same predictors when ran as part of the MAKER2 gene annotation pipeline.

Holt and Yandell *BMC Bioinformatics* 2011 **12**:491 doi:10.1186/1471-2105-12-491

Comparison of gene accuracies for MAKER vs. *ab initio* gene predictors

What about emerging model organisms for which little data is available? Gene prediction in classic model organisms is relatively simple because there are already a large number of experimentally determined and verified gene models, but with emerging model organisms, we are lucky to have a handful of gene models to train with. As a result *ab initio* gene predictors generally perform very poorly on emerging genomes.



MAKER's performance on the *S. mediterranea* emerging model organism genome. Pfam domain content of gene models determined using rpsblast

By using *ab inito* gene predictors within the MAKER pipeline you get several key benefits:

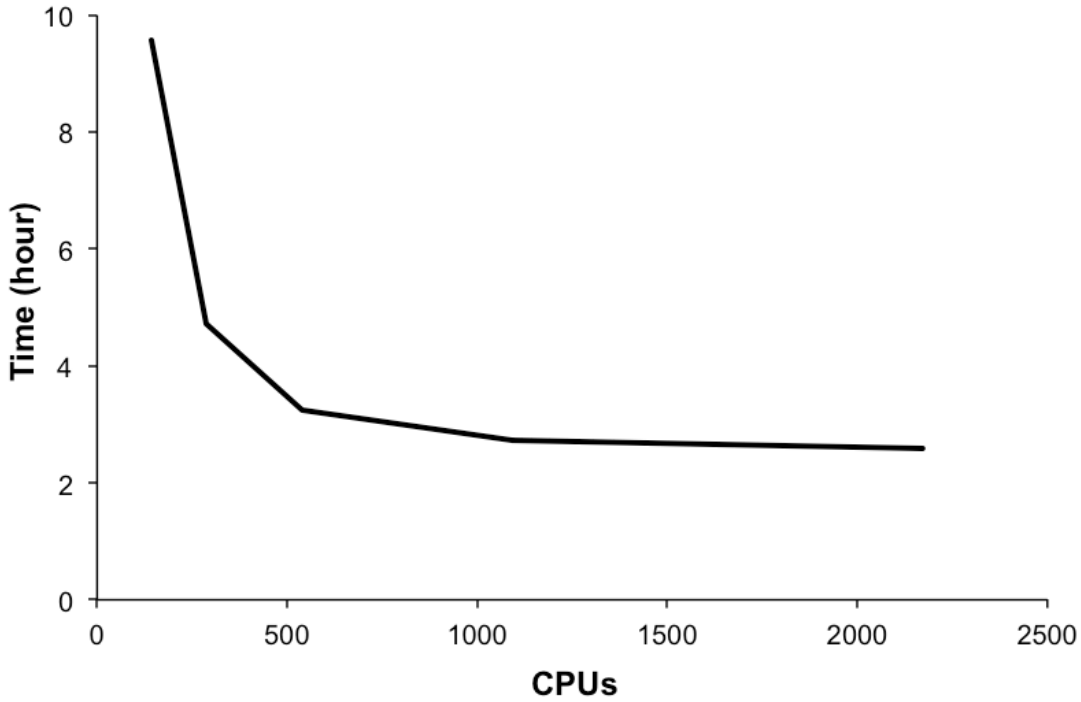
- MAKER provides gene models together with an evidence trail - useful for manual curation and quality control.
- MAKER provides a framework within which you can train and retrain gene predictors for improved performance.
- MAKER's output (including supporting evidence) can easily be loaded into a GMOD compatible database for annotation distribution.
- MAKER's annotations can be easily updated with new evidence by passing existing annotation sets back though MAKER.

MAKER for Plant Genomes

Plants are notoriously hard annotation targets. Plant genomes are commonly large and highly repetitive; they contain a large number of pseudogenes, and novel protein coding and non-coding genes. To address these challenges we optimized MAKER's performance on large computing clusters such at TACC, developed tutorials for custom repeat library generation, provide a pseudogene identification protocol for use with standard MAKER outputs, and incorporated non-coding RNA annotation capabilities into MAKER.

Performance on large computing clusters

MAKER is built with MPI support and plays well with OpenMPI, MPICH2, and Intel MPI. MAKER is preinstalled and available on some XSEDE clusters (a program funded by the US National Science Foundation). Through XSEDE, US based researchers and their international collaborators can request allocations on large computer clusters across the United States. As shown below, using the Lonestar cluster at the Texas Advanced Computing Center (TACC), the entire maize v2 genome (~2 Gb) could be annotated in just over 2 hours on ~500 cpus. XSEDE resources were also used to annotate the second largest genome ever sequenced (loblolly pine, >20 Gb) in less than 15 hours runtime on 8,640 cpus (37 hours total time when including queue wait time).



MAKER run times on the maize V2 genome assembly versus number of processors used

Tutorials for custom repeat library generation

Many of the interesting genomes we are currently sequencing as a genomics community are not being sequenced because of their similarities to previously sequenced genomes but because of their dis-similarities. These phylogenetically distant organisms not only present unique protein coding genes but also a multitude of previously unseen repetitive elements. For the best annotation results a species specific repeat library should be used in masking the genome prior to annotation. We have provided basic Repeat Library Construction--Basic and advanced Repeat Library Construction--Advanced tutorials for creating these libraries. A pipeline that automates this process is currently in development.

Pseudogene identification

MAKER does not identify pseudogenes directly but we do supply a separate pseudogene identification protocol that identifies potential pseudogenes as intergenic sequences with significant resemblance to annotated proteins in that genome. instructions for running that pipeline can be found here Protocol:Pseudogene (http://shiulab.plantbiology.msu.edu/wiki/index.php/Protocol:Pseudogene). This protocol can be adapted to find pseudogenes without similarity to protein coding genes in the organism but similar to genes in closely related species by modifying the input sequences to the pipeline.

Non-coding RNA annotation

tRNAscan-SE (http://selab.janelia.org/software.html) and snoscan (http://lowelab.ucsc.edu/snoscan/) are now integrated into the MAKER framework. Annotating tRNAs is now as simple as setting a single option in the maker_opts.ctl file. tRNAscan-SE runs quickly and accurately. Annotating snoRNAs requires the user to pass a file containing annotated rRNAs for the organism of interest in fasta format to MAKER through the maker_opts.ctl file. Currently all snoscan annotations are being promoted to the final annotation set. To increase specificity and overall accuracy, a filter based on AED will soon be implemented. miR-PREFeR (https://github.com/hangelwen/miR-PREFeR) was developed for miRNA annotation as part of the MAKER tool kit and has yet to be incorporated into the MAKER framework. At this time miR-PREFeR is run as a stand-alone tool and the output can be passed to MAKER in the maker_opts.ctl as 'other_gff=' for inclusion in the final gff3 file.

Data for Tutorial Examples

To get started we need to load some files in your home directory that we will use for all examples today. The files are in a tarball in the class directory already on the server, but can also be downloaded here (http://weatherby.genetics.utah.edu/data/maker_tutorial.tgz).

```
cd ~/
tar -zxf /gdc_home5/groups/tripleA_2018/3_wednesday/maker_tutorial.tgz
cd maker_tutorial
ls -1
```

You should see several example folders.

```
example_01_basic
example_02_abinitio
example_03_legacy
example_04_postannotation
example_05_yeast
```

Let's look inside example_01

```
ls -1 example_01_basic

dpp_contig.fasta
dpp_est.fasta
dpp_protein.fasta
finished1.tgz
finished2.tgz
hsap_contig.fasta
hsap_est.fasta
hsap_protein.fasta
opts1.txt
opts2.txt
```

You will see files called `opts.txt` and other files called `finished.tgz`. The finished.tgz files contains much of the final results for an example (think of it as the pre-baked food in a cooking show and the opts.txt file is a backup copy of the MAKER control file that we will be generating (more detail in a minute). Each of the other examples will contain similar pre-baked results files and control files so we don't have to wait for long running processes to complete.

Now let's get started!

Installation

Prerequisites

Perl Modules

- BioPerl (http://search.cpan.org/~cjfields/BioPerl/BioPerl.pm)
- DBI (http://search.cpan.org/~timb/DBI/DBI.pm)
- File::Which (http://search.cpan.org/~adamk/File-Which/lib/File/Which.pm)
- Perl::Unsafe::Signals (http://search.cpan.org/~rgarcia/Perl-Unsafe-Signals/Signals.pm)
- Bit::Vector (http://search.cpan.org/~stbey/Bit-Vector/Vector.pod)
- Inline (http://search.cpan.org/~sisyphus/Inline/Inline.pod)
- Inline::C (http://search.cpan.org/~sisyphus/Inline/C/C.pod)
- forks (http://search.cpan.org/~rybskej/forks/lib/forks.pm)
- forks::shared (http://search.cpan.org/~rybskej/forks/lib/forks/shared.pm)
- IO::All (http://search.cpan.org/~ingy/IO-All/lib/IO/All.pod)(Optional, for accessory scripts)
- IO::Prompt (http://search.cpan.org/~dconway/IO-Prompt/lib/IO/Prompt.pm)(Optional, for accessory scripts)

External Programs

- Perl (http://www.perl.org/) 5.8.8 or Higher
- SNAP (http://korflab.ucdavis.edu/software.html) version 2009-02-03 or higher
- RepeatMasker (http://www.repeatmasker.org/) 3.1.6 or higher
- Exonerate (https://www.ebi.ac.uk/about/vertebrate-genomics/software/exonerate) 1.4 or higher
- NCBI BLAST (http://www.ncbi.nlm.nih.gov/Ftp/) 2.2.X or higher

Optional Components:

- Augustus (http://bioinf.uni-greifswald.de/augustus/) 2.0 or higher
- GeneMark-ES (http://exon.biology.gatech.edu/) 2.3a or higher
- FGENESH (http://www.softberry.com/) 2.6 or higher

Required for optional MPI support:

- OpenMPI (http://www.open-mpi.org)
- MPICH (http://www.mpich.org)

The MAKER Package

MAKER can be downloaded from:

- http://www.yandell-lab.org/ - but it should already be installed on the servers

Now let's look at our MAKER installation:

```
cd ~/maker_tutorial/maker
ls -1
```

Note: That is a *dash one*, not a *dash L*, on the `ls` command.

You should now see the following:

```
data
GMOD
INSTALL
lib
LICENSE
MWAS
README
RELEASE
src
```

There are two files in particular that you would want to look at when installing MAKER - `INSTALL` and `README`. `INSTALL` gives a brief overview of MAKER and prerequisite installation. Let's take a look at this.

```
less INSTALL
```

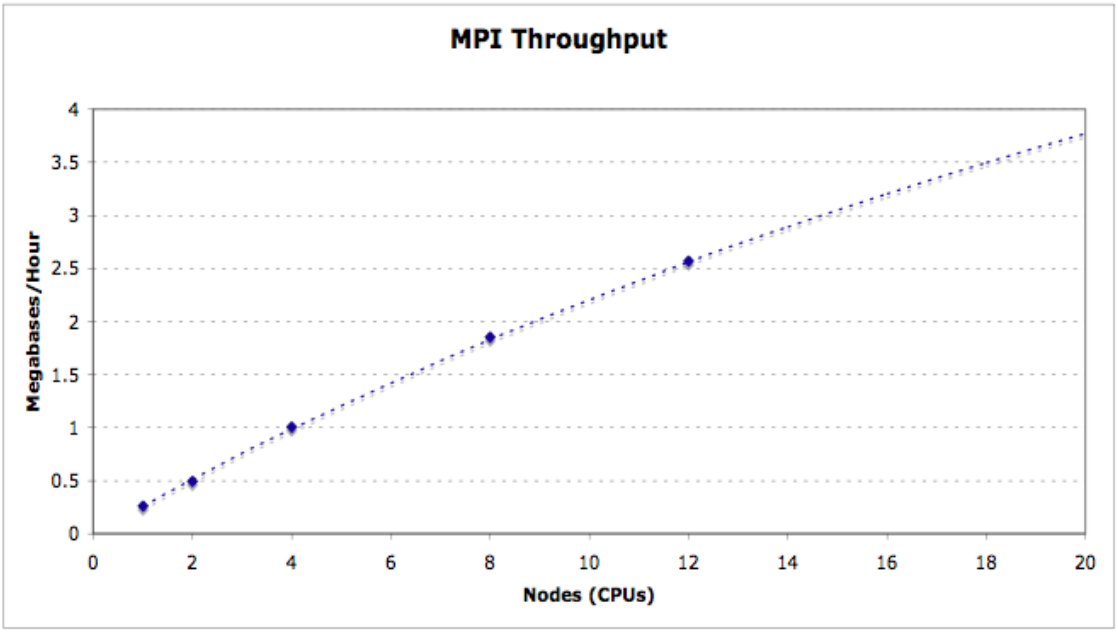
You really only need to follow the instructions for the EASY INSTALL, unless you have a problem/error during installation. Then you can follow the detailed install instructions in the file.

```
***Installation Documentation***

How to Install Standard MAKER

!!IMPORTANT NOTE FOR MAC OS X USERS!!
You will need to install developer tools (i.e. Xcode) from the App Store or
your installation disk. Also install fink (http://www.finkproject.org/) and
then install glib2-dev via fink (i.e. 'fink install glib2-dev').


**EASY INSTALL

1.  Go to the .../maker/src/ directory and run 'perl Build.PL' to configure.

2.  Accept default configuration options by just pressing enter. See MPI INSTALL
    in next section if you decide to configure for MPI.

3.  type './Build install' to complete the installation.

4.  If anything fails, either use the ./Build file commands to retry the failed
    section (i.e. './Build installdeps' and './Build installexes') or follow the
```

```
   detailed install instructions in the next section to manually install missing
   modules or programs. Use ./Build status to see available commands.

      ./Build status           #Shows a status menu
      ./Build installdeps      #installs missing PERL dependencies
      ./Build installexes      #installs missing external programs
      ./Build install          #installs MAKER

   Note: You do not need to be root.  Just say 'yes' to 'local install' when
   running './Build installdeps' and dependencies will be installed under
   .../maker/perl/lib, also missing external tools will be installed under
   .../maker/exe when running './Build installexes'.

   Note: For failed auto-download of external tools, when using the command
   './Build installexes', the .../maker/src/locations file is used to identify
   download URLs. You can edit this file to point to any alternate locations.
```

MPI Support

MAKER optionally supports Message Passing Interface (MPI), a parallel computation communication protocol primarily used on computer clusters. This allows MAKER jobs to be broken up across multiple nodes/processors for increased performance and scalability.



To use this feature, you must have MPICH2 installed with the the `--enable-sharedlibs` flag set during installation (See MPICH2 Installer's Guide). Or you can use OpenMPI, but you must preload shared libraries by adding a line like this to your ~/.bash_profile --> `export LD_PRELOAD=/usr/lib64/openmpi-1.10/lib/libmpi.so`

I may show some examples of using MPI.

Install example:

```
module load openmpi-1.10-x86_64
cd ~/maker_tutorial/maker/src
perl Build.PL
```

```
MAKER supports distributed parallelization via MPI.
Would you like to configure MAKER for MPI (This
requires that you have an MPI client installed)? [N ]y

Please specify the path to 'mpicc' on your system: [/usr/lib64/openmpi-1.10/bin/mpicc ]
/usr/lib64/openmpi-1.10/bin/mpicc

Please specify the path to the directory containing 'mpi.h': [/usr/include/mpich-x86_64 ] /usr/include/openmpi-1.10-
x86_64/


./Build install
```

Now after install, if you look inside the base MAKER directory again, you will see two new folders (.../bin/ and .../perl/). The 'bin' directory contains the executables for MAKER and the 'perl' contains compiled libraries built just now for the MAKER installation.

You should now see the following:

```
cd ~/maker_tutorial/maker
ls -1

 bin
 data
 GMOD
 INSTALL
 lib
 LICENSE
 MWAS
 perl
 README
 RELEASE
 src

ls -1 bin

 cegma2zff
 chado2gff3
 compare
 cufflinks2gff3
 evaluator
 fasta_merge
 fasta_tool
 genemark_gtf2gff3
 gff3_merge
 iprscan2gff3
 iprscan_wrap
 ipr_update_gff
 maker
 maker2chado
 maker2eval_gtf
 maker2jbrowse
 maker2wap
 maker2zff
 maker_functional
 maker_functional_fasta
 maker_functional_gff
 maker_map_ids
 map2assembly
 map_data_ids
 map_fasta_ids
 map_gff_ids
 mpi_evaluator
 mpi_iprscan
 tophat2gff3
```

Getting Started with MAKER

First let's test our MAKER executable and look at the usage statement:

```
maker -h
```

```
MAKER version 2.31.9

Usage:

     maker [options] <maker_opts> <maker_bopts> <maker_exe>


Description:

     MAKER is a program that produces gene annotations in GFF3 format using
     evidence such as EST alignments and protein homology. MAKER can be used to
     produce gene annotations for new genomes as well as update annotations
     from existing genome databases.

     The three input arguments are control files that specify how MAKER should
     behave. All options for MAKER should be set in the control files, but a
     few can also be set on the command line. Command line options provide a
     convenient machanism to override commonly altered control file values.
     MAKER will automatically search for the control files in the current
     working directory if they are not specified on the command line.

     Input files listed in the control options files must be in fasta format
     unless otherwise specified. Please see MAKER documentation to learn more
     about control file  configuration.  MAKER will automatically try and
     locate the user control files in the current working directory if these
     arguments are not supplied when initializing MAKER.

     It is important to note that MAKER does not try and recalculated data that
     it has already calculated.  For example, if you run an analysis twice on
     the same dataset you will notice that MAKER does not rerun any of the
     BLAST analyses, but instead uses the blast analyses stored from the
     previous run. To force MAKER to rerun all analyses, use the -f flag.

     MAKER also supports parallelization via MPI on computer clusters. Just
     launch MAKER via mpiexec (i.e. mpiexec -n 40 maker). MPI support must be
     configured during the MAKER installation process for this to work though


Options:

     -genome|g <file>    Overrides the genome file path in the control files

     -RM_off|R           Turns all repeat masking options off.

     -datastore/         Forcably turn on/off MAKER's two deep directory
      nodatastore        structure for output.  Always on by default.

     -old_struct         Use the old directory styles (MAKER 2.26 and lower)

     -base    <string>   Set the base name MAKER uses to save output files.
                         MAKER uses the input genome file name by default.

     -tries|t <integer>  Run contigs up to the specified number of tries.

     -cpus|c  <integer>  Tells how many cpus to use for BLAST analysis.
                         Note: this is for BLAST and not for MPI!

     -force|f            Forces MAKER to delete old files before running again.
                         This will require all blast analyses to be rerun.

     -again|a            recaculate all annotations and output files even if no
                         settings have changed. Does not delete old analyses.

     -quiet|q            Regular quiet. Only a handlful of status messages.

     -qq                 Even more quiet. There are no status messages.

     -dsindex            Quickly generate datastore index file. Note that this
```

```
                        will not check if run settings have changed on contigs

    -nolock             Turn off file locks. May be usful on some file systems,
                        but can cause race conditions if running in parallel.

    -TMP                Specify temporary directory to use.

    -CTL                Generate empty control files in the current directory.

    -OPTS               Generates just the maker_opts.ctl file.

    -BOPTS              Generates just the maker_bopts.ctl file.

    -EXE                Generates just the maker_exe.ctl file.

    -MWAS    <option>   Easy way to control mwas_server for web-based GUI

                             options:  STOP
                                       START
                                       RESTART

    -version            Prints the MAKER version.

    -help|?             Prints this usage statement.
```

Running MAKER with example data

When you install, MAKER it comes with some example input files to test the installation and to familiarize the user with how to run the pipline. The example files are found in the `.../maker/data` directory.

```
ls -1 /usr/local/maker/data

 dpp_contig.fasta
 dpp_est.fasta
 dpp_protein.fasta
 hsap_contig.fasta
 hsap_est.fasta
 hsap_protein.fasta
 te_proteins.fasta
```

The example files are in FASTA format. MAKER requires FASTA format for its input files. Let's take a look at one of theses files to see what the format looks like.

```
less /usr/local/maker/data/dpp_protein.fasta


>dpp-CDS-1
MRAWLLLLAVLATFQTIVRVASTEDISQRFIAAIAPVAAHIPLASASGSGSGRSGSRSVG
ASTSTALAKAFNPFSEPASFSDSDKSHRSKTNKKPSKSDANRQFNEVHKPRTDQLENSKN
KSKQLVNKPNHNKMAVKEQRSHHKKSHHHRSHQPKQASASTESHQSSSIESIFVEEPTLV
LDREVASINVPANAKAIIAEQGPSTYSKEALIKDKLKPDPSTLVEIEKSLLSLFNMKRPP
KIDRSKIIIPEPMKKLYAEIMGHELDSVNIPKPGLLTKSANTVRSFTHKDSKIDDRFPHH
HRFRLHFDVKSIPADEKLKAAELQLTRDALSQQVVASRSSANRTRYQVLVYDITRVGVRG
QREPSYLLLDTKTVRLNSTDTVSLDVQPAVDRWLASPQRNYGLLVEVRTVRSLKPAPHHH
VRLRRSADEAHERWQHKQPLLFTYTDDGRHKARSIRDVSGGEGGGKGGRNKRQPRRPTRR
KNHDDTCRRHSLYVDFSDVGWDDWIVAPLGYDAYYCHGKCPFPLADHFNSTNHAVVQTLV
NNMNPGKVPKACCVPTQLDSVAMLYLNDQSTVVLKNYQEMTVVGCGCR
```

FASTA format is fairly simple. It contains a definition line starting with '>' that contains a name for a sequence followed by the actual nucleotide or amino acid sequence on subsequent lines. The file we are looking at contains protein sequences, so the sequence uses the single letter code for amino acids.

A minimal input file set for MAKER would generally consist of a FASTA file for the genomic sequence, a FASTA file of RNA (ESTs/cDNA/mRNA transcripts) from the organism, and a FASTA file of protein sequences from the same or related organisms (or a general protein database).

I've already copied these data files into the `~/maker/tutorial/example_01_basic` directory for you, but if you're following this tutorial outside of the course you can run directly inside the data directory to follow the first example or copy the files into a directory of your choice:

Now let's move to the first example.

```
cd ~/maker_tutorial/example_01_basic
```

As you can see it contains the same files as the `data/` directory that comes with MAKER.

```
ls -1

dpp_contig.fasta
dpp_est.fasta
dpp_protein.fasta
finished1.tgz
finished2.tgz
hsap_contig.fasta
hsap_est.fasta
hsap_protein.fasta
opts1.txt
opts2.txt
```

Next we need to tell MAKER all the details about how we want the annotation process to proceed. Because there can be many variables and options involved in annotation, command line options would be too numerous and cumbersome. Instead MAKER uses a set of configuration files which guide each run. You can create a set of generic configuration files in the current working directory by typing the following.

```
maker -CTL
```

This creates three files (type `ls -1` to see).

- `maker_exe.ctl` - contains the path information for the underlying executables.
- `maker_bopt.ctl` - contains filtering statistics for BLAST and Exonerate
- `maker_opt.ctl` - contains all other information for MAKER, including the location of the input genome file.

Control files are run-specific and a separate set of control files will need to be generated for each genome annotated with MAKER. MAKER will look for control files in the current working directory, so it is recommended that MAKER be run in a separate directory containing unique control files for each genome.

Let's take a look at the `maker_exe.ctl` file (here we use nano but you can use any text editor you want).

```
nano maker_exe.ctl
```

You will see the names of a number of MAKER supported executables as well as the path to their location. If you followed the installation instructions correctly, including the instructions for installing prerequisite programs, all executable paths should show up automatically for you. However if the location to any of the executables is not set in your PATH environment variable, as per installation instructions, you will have to add these manually to the `maker_exe.ctl` file every time you run MAKER.

Lines in the MAKER control files have the format `key=value` with no spaces before or after the equals sign(=). If the value is a file name, you can use relative paths and environment variables, i.e. `snap=$HOME/snap`. Note that for all control files the comments written to help users begin with a pound sign(#). In addition, options before the equals sign(=) can not be changed, nor should there be a space before or after the equals sign.

Now let's take a look at the `maker_bopts.ctl` file.

```
nano maker_bopts.ctl
```

In this file you will find values you can edit for downstream filtering of BLAST and Exonerate alignments. At the very top of the file you will see that I have the option to tell MAKER whether I prefer to use WU-BLAST or NCBI-BLAST. We want to set this to NCBI-BLAST, since that is what is installed. We can just leave the remaining values as the default.

```
blast_type=ncbi+
```

Now let's take a look at the `maker_opts.ctl` file.

```
nano maker_opts.ctl
```

This is the primary configuration file for MAKER specific options. Here we need to set the location of the genome, EST, and protein input files we will be using. These come from the supplied example files. If you are following this in class you can replace the `maker_opts.ctl` file with the `opts.txt` which is has options pre-filled for you. There are a lot of options in this file, and we'll discuss many of them in more detail later on in other examples. Below are the options we adjust with a text editor:

```
genome=dpp_contig.fasta
est=dpp_est.fasta
protein=dpp_protein.fasta
est2genome=1

    or

cp opts1.txt maker_opts.ctl
```

Note: You should not put spaces on either side of the = on the above control file lines.

Now let's run MAKER.

```
maker
```

You should now see a large amount of status information flowing past your screen. If you don't want to see this you can run MAKER with the `-q` option for "quiet" on future runs.

Details of What is Going on Inside of MAKER

Repeat Masking

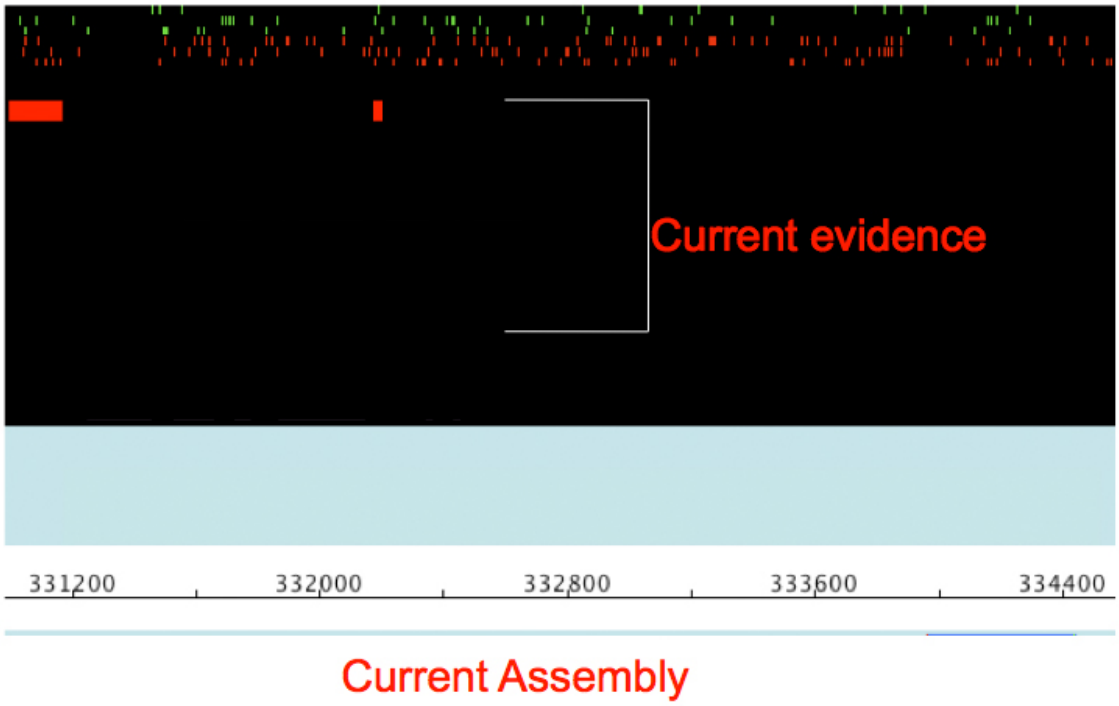
The first step in the MAKER pipeline is repeat masking. Why do we need to do this? Repetitive elements can make up a significant portion of the genome. These repeats fall into to basic classes:

1. Low-complexity (simple) repeats: These consist of stretches (sometimes very long) of tandemly repeated sequences with little information content. Examples of low-complexity sequence are mononucleotide runs (AAAAAAA, GGGGGG) and the various types of satellite DNA.
2. Interspersed (complex) repeats - Sections of sequence that have the ability to change thier location within the genome. These transposons and retrotransposons contain real coding genes (reverse transcriptase, Gag, Pol) and have the ability to transpose (and often duplicate) surrounding sequence with them.

The low information content of the low complexity repeats sequence can produce sequence alignments with high statistical significance to low-complexity protein regions creating a false homology (think evidence for genes) throughout the genome.

Because these complex repeats contain real protein coding genes they play havoc with *ab initio* gene predictors. For example, a transposable element that occurs within the intron of one of the organism's own protein encoding genes might cause a gene predictor to include extra exons as part of this gene. Thus, sequence which really only belongs to a transposable element is included in your final gene annotation set.

Analysis of the repeat structure of a new genome is an important goal, but the presence of those repeats both simple and complex makes it nearly impossible to generate a useful annotation set of the organisms own genes. For this reason it is critical to identify and mask these repetitive regions of the genome.



Identify and mask repetitive elements

MAKER identifies repeats in two steps.

- First MAKER runs a program called RepeatMasker is used to identify both all classes of repeats that match entries in the RepBase repeat library. You can even create your own species specific repeat library and RepeatMasker will use

it in addition to its own libraries to mask repeats. Species specific repeat libraries can improve the annotation tremendously instructions for creating aa repeat library for your favorite organism can be found here Repeat Library Construction--Basic and here Repeat Library Construction--Advanced.

- Next MAKER uses RepeatRunner to identify transposable elements and viral proteins using the RepeatRunner protein database. Because RepeatRunner uses protein sequence libraries and protein sequence diverges at a slower rate than nucleotide sequence, this step picks up many problematic regions of divergent repeats that are missed by RepeatMasker (which searches in nucleotide space).

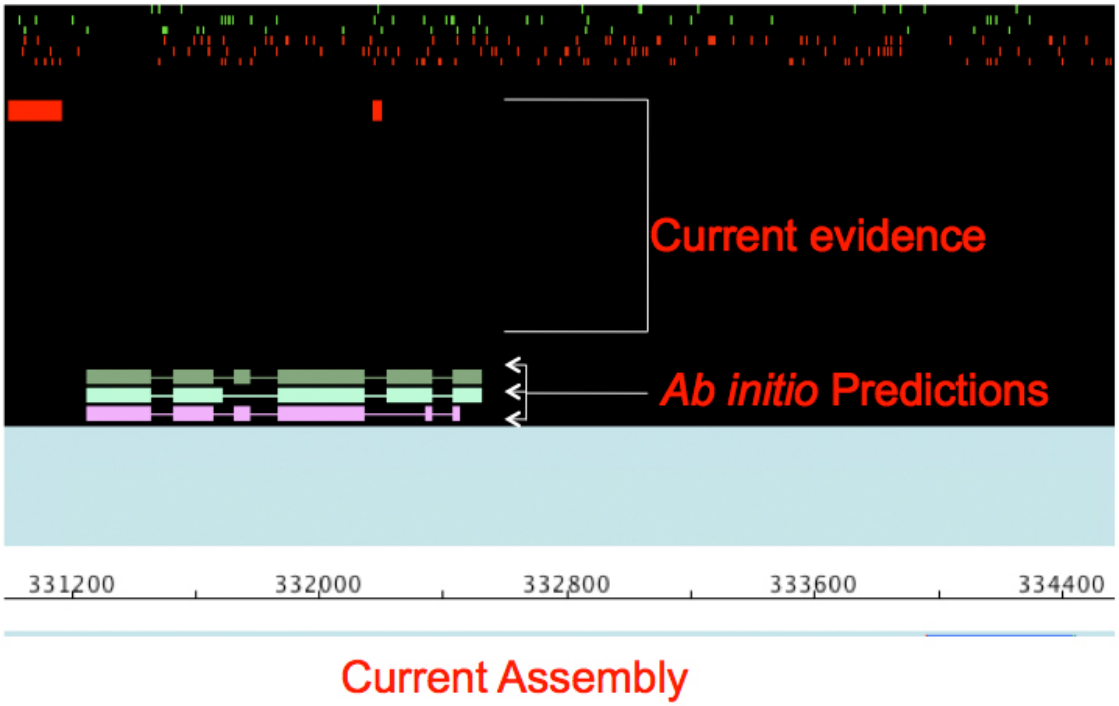
Regions identified during repeat analysis are masked out in two different ways:

1. Complex repeats are hard-masked - the repeat sequence is replaced with the letter N. This essentially removes this sequence from any further consideration at any later point of the annotation process.
2. Simple repeats are soft-masked - sequences are transformed to lower case. This prevents alignment programs such as Blast from seeding any new alignments in the soft-masked region, however alignments that begin in a nearby (non-masked) region of the genome can extend into the soft-masked region. This is important because low-complexity regions are found within many real genes, they just don't make up the majority of the gene.

Masking sequence from the annotation pipeline (especially hard masking) may seem like it might cause us to lose real protein coding genes that are important for the organism's biology. It is true that repeat derived genes can be co-opted and expressed by the organism and repeat masking will affect our ability to annotate these genes. However, these genes are rare and the number of gene models and sequence alignments improved by the repeat masking step far outweighs the few gene models that may be negatively affected. You do have the option to run *ab initio* gene predictors on both the masked and unmasked sequence if repeat masking worries you though. You do this by setting unmask:1 in the `maker_opt.ctl` configuration file.

Ab Initio Gene Prediction

Following repeat masking, MAKER runs *ab initio* gene predictors specified by the user to produce preliminary gene models. *Ab initio* gene predictors produce gene predictions based on underlying mathematical models describing patterns of intron/exon structure and consensus start signals. Because the patterns of gene structure are going to differ from organism to organism, you must train gene predictors before you can use them. I will discuss how to do this later on.



MAKER currently supports:

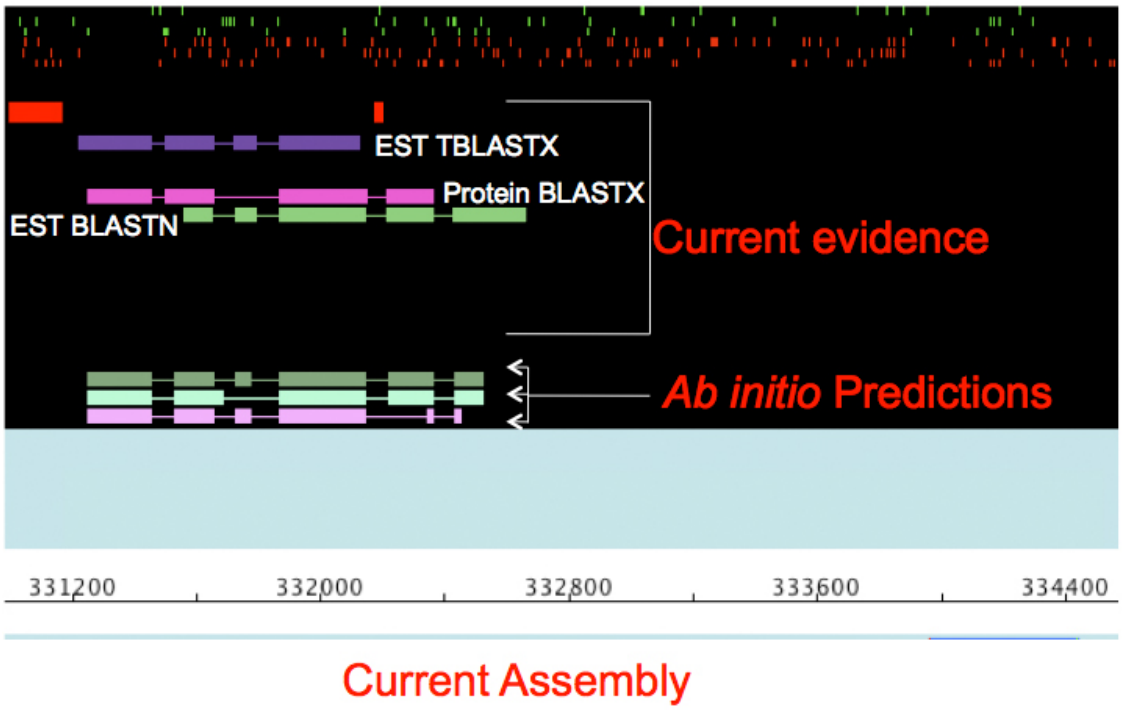
- SNAP (Works good, easy to train, not as good as others on longer intron genomes).
- Augustus (Works great, hard to train, but getting better)
- GeneMark (Self training, MAKER doesn't support hints for GeneMark, not good for fragmented genomes or long introns).
- FGENESH (Works great, costs money even for training)

You must specify in the maker_opts.ctl file the training parameters file you want to use use when running each of these algorithms.

RNA and Protein Evidence Alignment

A simple way to indicate if a sequence region is likely associated with a gene is to identify (A) if the region is actively being transcribed or (B) if the region has homology to a known protein. This can be done by aligning Expressed Sequence Tags (ESTs) and proteins to the genome using alignment algorithms.

- ESTs are sequences derived from a cDNA library. In recent years ESTs have been largely replaced by mRNA-seq data, which have decreases costs but have may of same challenges as traditional EST libraries. Because of the difficulties associated with working with mRNA and depending on how the cDNA library was prepared, EST databases and mRNA-seq assemblies usually represent bits and pieces of transcribed RNA with only a few full length transcripts. MAKER aligns these sequences to the genome using BLASTN. If ESTs/mrNA-seq from the organism being annotated are unavailable or sparse, you can use ESTs/mRNA-seq from a closely related organism. However, RNA from closely related organisms are unlikely to align using BLASTN since nucleotide sequences can diverge quite rapidly. For these RNAs, MAKER uses TBLASTX to align them in protein space.
- Protein sequence generally diverges quite slowly over large evolutionary distances, as a result proteins from even evolutionarily distant organisms can be aligned against raw genomic sequence to try and identify regions of homology. MAKER does this using BLASTX.



Align EST and protein evidence

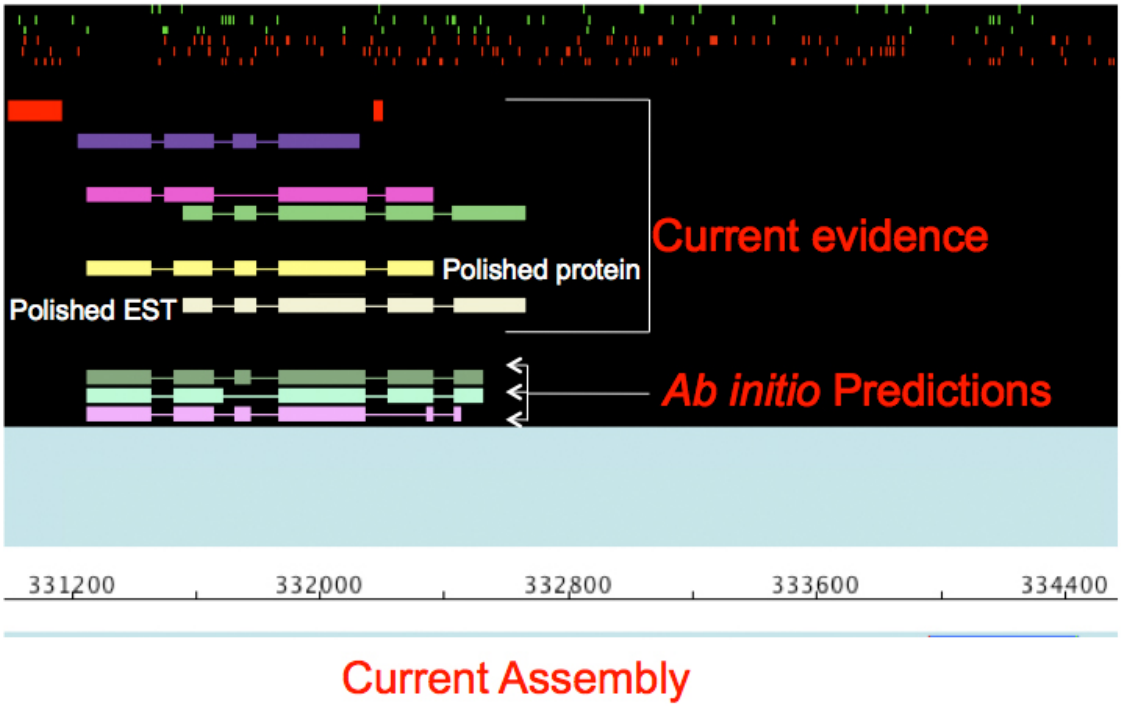
Remember now that we are aligning against the repeat-masked genomic sequence. How is this going to affect our alignments? For one thing we won't be able to align against low-complexity regions. Some real proteins contain low-complexity regions and it would be nice to identify those, but if I let anything align to a low-complexity region, then I will get spurious alignments all over the genome. Wouldn't it be nice if there was a way to allow BLAST to extend alignments through low-complexity regions, but only if there is is already alignment somewhere else? You can do this with soft-

masking. If you remember soft-masking is using lower case letters to mask sequence without losing the sequence information. BLAST allows you to use soft-masking to keep alignments from seeding in low-complexity regions, but allows you to extend through them. This of course will allow some of the spurious alignments you were trying to avoid, but overall you still end up suppressing the majority of poor alignments while letting through enough real alignments to justify the cost. You can turn this behavior off though if it bothers you by setting `softmask=0` in the `maker_bopt.ctl` file.

Polishing Evidence Alignments

Because of oddities associated with how BLAST statistics work, BLAST alignments are not as informative as they could be. BLAST will align regions any where it can, even if the algorithm aligns regions out of order, with multiple overlapping alignments in the exact same region, or with slight overhangs around splice sites.

To get more informative alignments MAKER uses the program Exonerate to polish BLAST hits. Exonerate realigns each sequences identified by BLAST around splice sites and forces the alignments to occur in order. The result is a high quality alignment that can be used to suggest near exact intron/exon positions. Polished alignments are produced using the est2genome and protein2genome options for Exonerate.

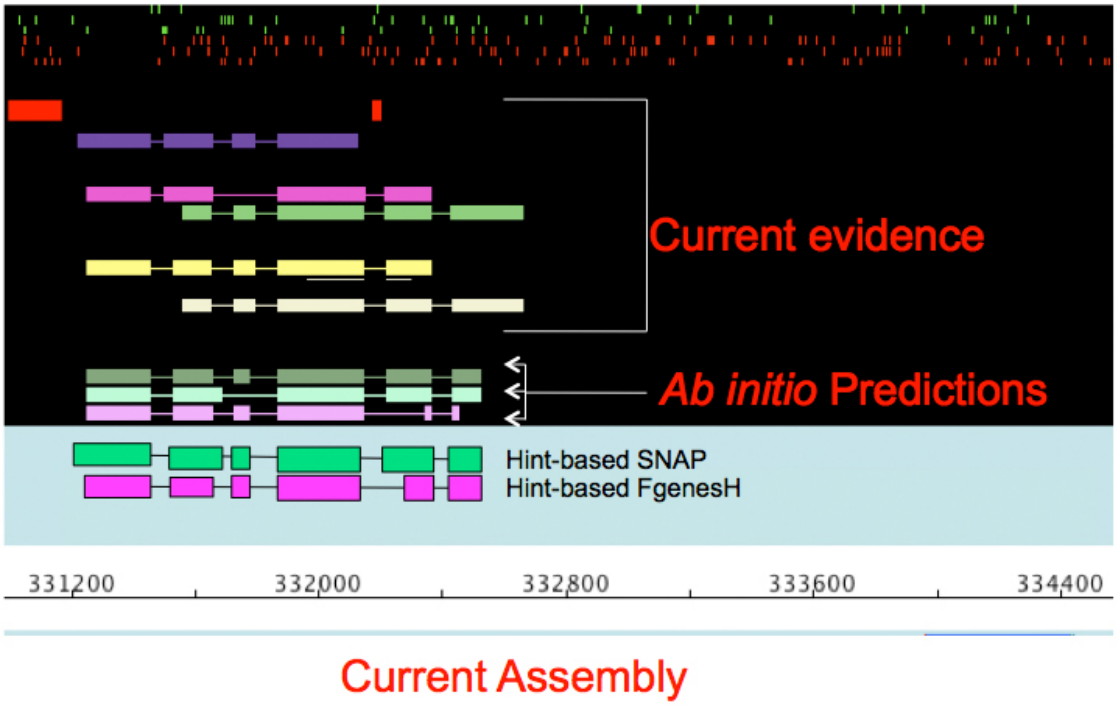


Polish BLAST alignments with Exonerate

One of the benefits of polishing EST alignments is the ability to identify the strand an EST derives from. Because of amplification steps involved in building an EST library and limitations involved in some high throughput sequencing technologies, you don't necessarily know whether you're really aligning the forward or reverse transcript of an mRNA. However, if you take splice sites into account, you can only align to one strand correctly.

Integrating Evidence to Synthesize Annotations

Once you have *ab initio* predictions, EST alignments, and protein alignments you can integrate this evidence to produce even better gene predictions. MAKER does this by communicating with the gene prediction programs. MAKER takes all the evidence, generates "hints" to where splice sites and protein coding regions are located, and then passes these "hints" to programs that will accept them.



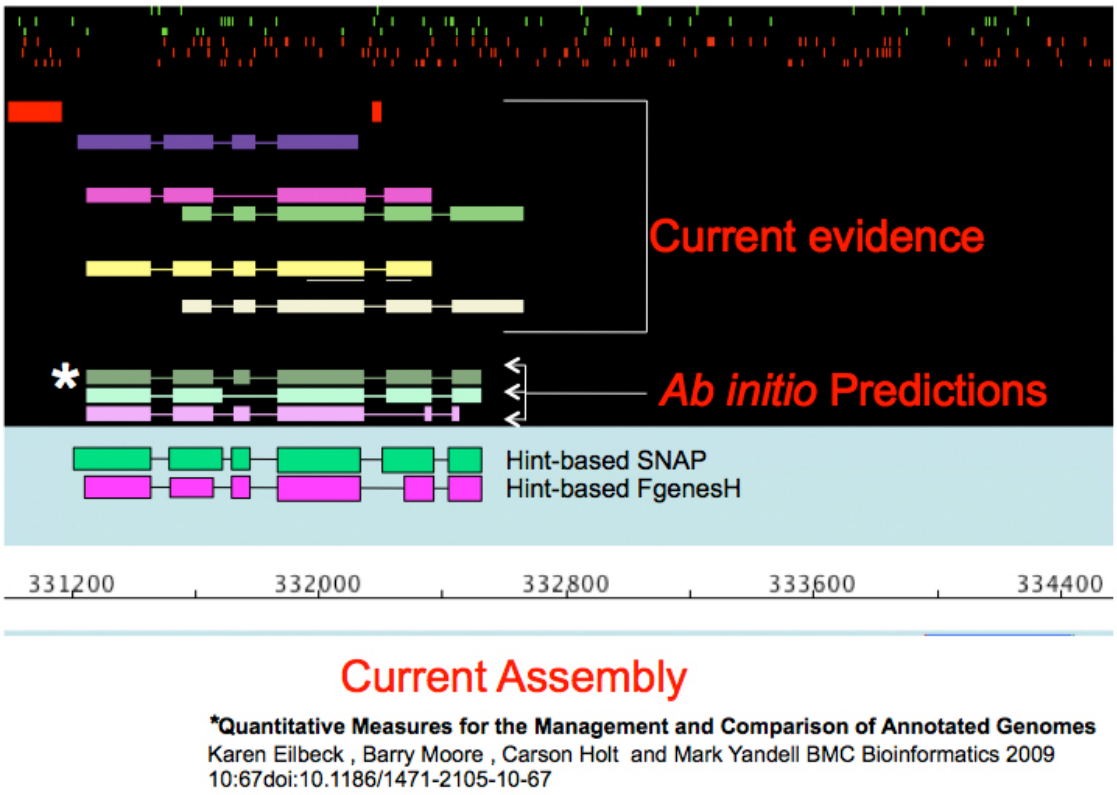
Pass gene finders evidence-based 'hints'

MAKER produces hint based predictors for:

- SNAP
- Augustus
- FGENESH
- GeneMark (under development)

Selecting and Revising the Final Gene Model

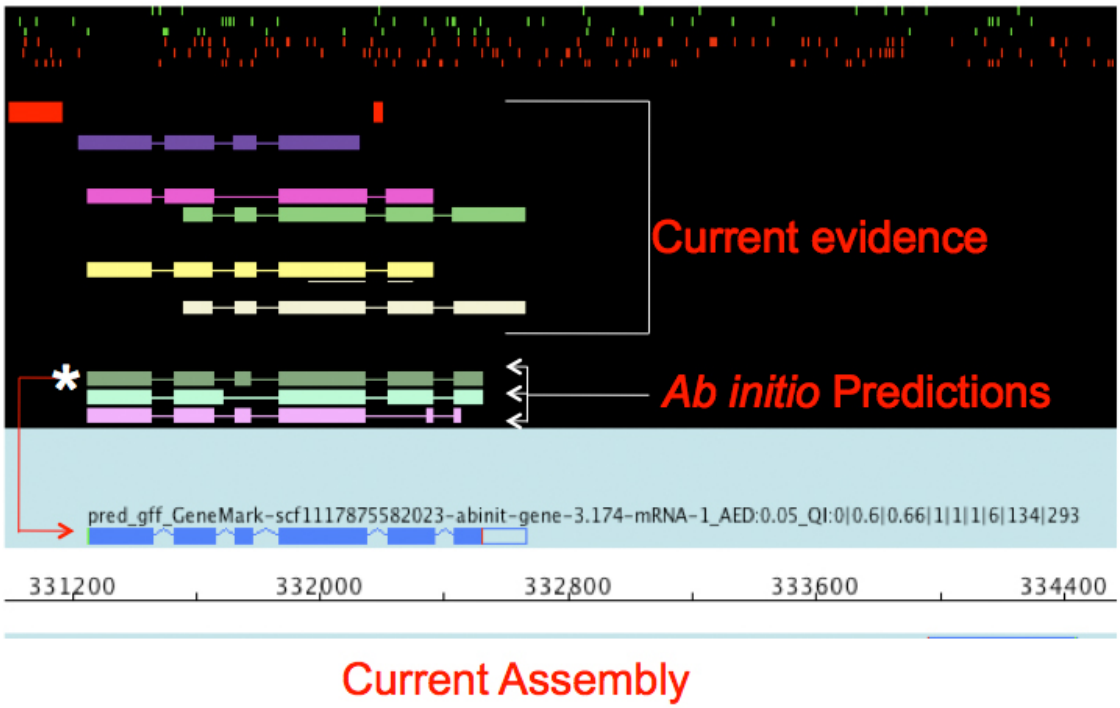
MAKER then takes the entire pool of *ab initio* and evidence informed gene predictions, updates features such as 5' and 3' UTRs based on EST evidence, tries to determine alternative splice forms where EST data permits, produces quality control metrics for each gene model (this is included in the output), and then MAKER chooses from among all the gene model possibilities the one that best matches the evidence. This is done using a modified sensitivity/specificity distance metric.



*Quantitative Measures for the Management and Comparison of Annotated Genomes
Karen Eilbeck , Barry Moore , Carson Holt and Mark Yandell BMC Bioinformatics 2009 10:67doi:10.1186/1471-2105-10-67

Identify gene model most consistent with evidence*

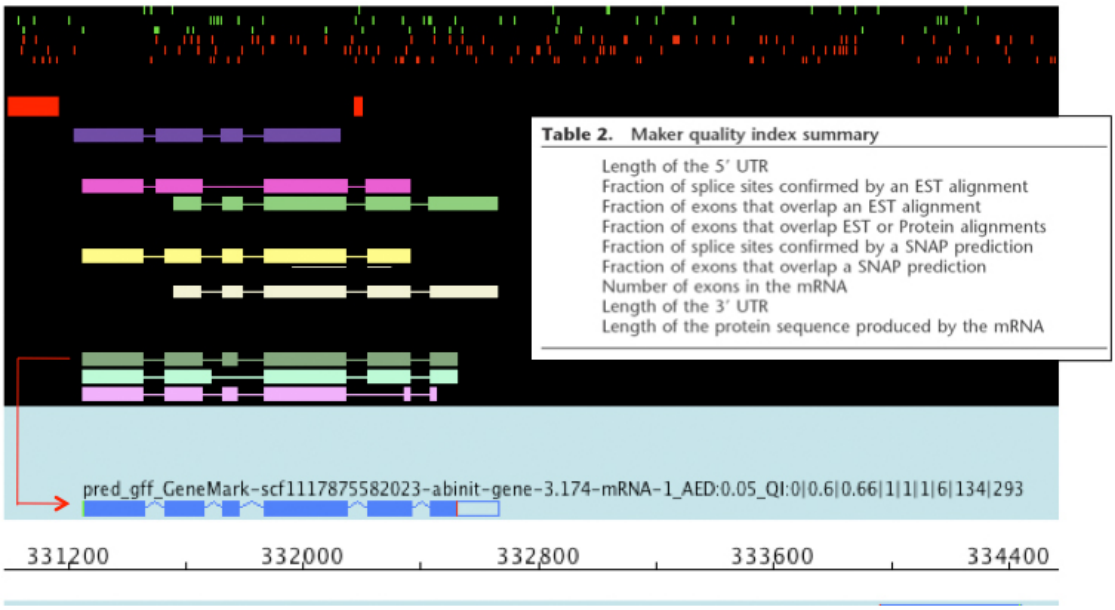
MAKER can use evidence from EST alignments to revise gene models to include features such as 5' and 3' UTRs.



Revise model further if necessary; create new annotation

Quality Control

Finally MAKER calculates quality control statistics to assist in downstream management and curation of gene models outside of MAKER.



Compute support for each portion of the gene model

MAKER's Output

If you look in the current working directory, you will see that MAKER has created an output directory called `dpp_contig.maker.output`. The name of the output directory is based on the input genomic sequence file, which in this case was `dpp_contig.fasta`.

Now let's see what's inside the output directory.

```
cd dpp_contig.maker.output
ls -1
```

You should now see a list of directories and files created by MAKER.

```
dpp_contig_datastore
dpp_contig_master_datastore_index.log
maker_bopts.log
maker_exe.log
maker_opts.log
mpi_blastdb
seen.dbm
```

- The `maker_opts.log`, `maker_exe.log`, and `maker_bopts.log` files are logs of the control files used for this run of MAKER.
- The `mpi_blastdb` directory contains FASTA indexes and BLAST database files created from the input EST, protein, and repeat databases.
- The `dpp_contig_master_datastore_index.log` contains information on both the run status of individual contigs and information on where individual contig data is stored.
- The `dpp_contig_datastore` directory contains a set of subfolders, each containing the final MAKER output for individual contigs from the genomic fasta file.

Once a MAKER run is finished the most important file to look at is the `dpp_contig_master_datastore_index.log` to see if there were any failures.

```
less -S dpp_contig_master_datastore_index.log
```

If everything proceeded correctly you should see the following:

```
contig-dpp-500-500      dpp_contig_datastore/05/1F/contig-dpp-500-500/  STARTED
contig-dpp-500-500      dpp_contig_datastore/05/1F/contig-dpp-500-500/  FINISHED
```

There are only entries describing a single contig because there was only one contig in the example file. These lines indicate that the contig `contig-dpp-500-500` STARTED and then FINISHED without incident. Other possible entries include:

- FAILED - indicates a failed run on this contig, MAKER will retry these
- RETRY - indicates that MAKER is retrying a contig that failed
- SKIPPED_SMALL - indicates the contig was too short to annotate (minimum contig length is specified in `maker_opt.ctl`)
- DIED_SKIPPED_PERMANENT - indicates a failed contig that MAKER will not attempt to retry (number of times to retry a contig is specified in `maker_opt.ctl`)

The entries in the `dpp_contig_master_datastore_index.log` file also indicate that the output files for this contig are stored in the directory `dpp_contig_datastore/contig-dpp-500-500/`. Knowing where the output is stored may seem trivial, however, input genome fasta files can contain thousands even hundreds-of-thousands of contigs, and many file-systems have performance problems with large numbers of sub-directories and files within a single directory. Even when the underlying file-system handles things gracefully, access via network file-systems can still be an issue. To deal with this problem, MAKER creates a hierarchy of nested sub-directory layers, starting from a 'base', and places the results for a given contig within these datastore of possibly thousands of nested directories. The `master_datastore_index.log` file this is essential for identifying where the output for a given contig is stored.

Now let's take a look at what MAKER produced for the contig 'contig-dpp-500-500'.

```
cd dpp_contig_datastore/05/1F/contig-dpp-500-500/
ls -1
```

The directory should contain a number of files and a directory.

```
contig-dpp-500-500.gff
contig-dpp-500-500.maker.proteins.fasta
contig-dpp-500-500.maker.transcripts.fasta
run.log
theVoid.contig-dpp-500-500
```

- The `contig-dpp-500-500.gff` contains all annotations and evidence alignments in GFF3 format (but just for a singel contig/scaffold). This is the important file for use with Apollo or GBrowse.
- The `contig-dpp-500-500.maker.transcripts.fasta` and `contig-dpp-500-500.maker.proteins.fasta` files contain the transcript and protein sequences for MAKER produced gene annotations on the contig.
- The `run.log` file is a log file. If you change settings and rerun MAKER on the same dataset, or if you are running a job on an entire genome and the system fails, this file lets MAKER know what analyses need to be deleted, rerun, or can be carried over from a previous run. One advantage of this is that rerunning MAKER is extremely fast, and your runs are virtually immune to all system failures.
- The directory `theVoid.contig-dpp-500-500` contains raw output files from all the programs MAKER runs (Blast, SNAP, RepeatMasker, etc.). You can usually ignore this directory and its contents.

The datastore directory contains one set of output files for each contig/chromosome from the input assembly, but at some point you're going to want merged files containing all of your output (i.e. a single GFF3 and FASTA file containing all genes). To do this we use two accessory scripts that come with MAKER: `gff3_merge` and `fasta_merge`. Both take the `master_datastore_index.log` file as input.

So let's move into the directory containing that file and run those scripts.

```
cd ~/maker_tutorial/example_01_basic/dpp_contig.maker.output

fasta_merge -d dpp_contig_master_datastore_index.log
gff3_merge -d dpp_contig_master_datastore_index.log
```

Now you will see a number of new files that represent the merged output for the entire assembly (in this case the assembly only contained a single contig though).

```
ls -1

dpp_contig.all.gff
dpp_contig.all.maker.proteins.fasta
dpp_contig.all.maker.transcripts.fasta
```

Viewing MAKER Annotations

Let's take a look at the GFF3 file produced by MAKER.

```
less -S dpp_contig.all.gff
```

As you can see, manually viewing the raw GFF3 file produced by MAKER really isn't that meaningful. While you can identify individual features such as genes, mRNAs, and exons, trying to interpret those features in the context of thousands of other genes and thousands of bases of sequence really can't be done by directly looking at the GFF3 file.

For sanity check purposes it would be nice to have a graphical view of what's in the GFF3 file. To do that, GFF3 files can be loaded into programs like Web Apollo, GBrowse, and JBrowse.

JBrowse

JBrowse ia convenient way to view and distribute MAKER GFF3 output, and it comes with a simple script called `maker2jbrowse` that makes loading MAKER's output into JBrowse extremely easy. I will demonstrate how to load a GFF3 into JBrowse, but for all the examples we do today, I've already provided links for viewing the results. So loading additional data into JBrowse will be an exercise left to the user outside of this tutorial. Below is a link to example 1.1 output loaded into JBrowse.

Click to see example 1.1 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example1.1/)

Advanced MAKER Configuration, Re-annotation Options, and Improving

Annotation Quality

The remainder of this page addresses issues that can be encountered during the annotation process. I then describe how MAKER can be used to resolve each issue.

Configuration Files in Detail

Let's take a closer look at the configuration options in the `maker_opt.ctl` file.

```
cd ~/maker_tutorial/example_01_basic
nano maker_opts.ctl
```

Genome Options (Required)

```
genome= #genome sequence (fasta file or fasta embeded in GFF3 file)
organism_type=eukaryotic #eukaryotic or prokaryotic. Default is eukaryotic
```

Re-annotation Using MAKER Derived GFF3

```
maker_gff= #MAKER derived GFF3 file
est_pass=0 #use ESTs in maker_gff: 1 = yes, 0 = no
altest_pass=0 #use alternate organism ESTs in maker_gff: 1 = yes, 0 = no
protein_pass=0 #use protein alignments in maker_gff: 1 = yes, 0 = no
rm_pass=0 #use repeats in maker_gff: 1 = yes, 0 = no
model_pass=0 #use gene models in maker_gff: 1 = yes, 0 = no
pred_pass=0 #use ab-initio predictions in maker_gff: 1 = yes, 0 = no
other_pass=0 #passthrough anyything else in maker_gff: 1 = yes, 0 = no
```

RNA/Transcript Evidence (the options are called EST for historic reasons)

```
est= #set of ESTs or assembled mRNA-seq in fasta format
altest= #EST/cDNA sequence file in fasta format from an alternate organism
est_gff= #aligned ESTs or mRNA-seq from an external GFF3 file
altest_gff= #aligned ESTs from a closly relate species in GFF3 format
```

Protein Homology Evidence

```
protein= #protein sequence file in fasta format (i.e. from mutiple oransisms)
protein_gff=  #aligned protein homology evidence from an external GFF3 file
```

Repeat Masking

```
model_org=all #select a model organism for RepBase masking in RepeatMasker
rmlib= #provide an organism specific repeat library in fasta format for RepeatMasker
repeat_protein= #provide a fasta file of transposable element proteins for RepeatRunner
rm_gff= #pre-identified repeat elements from an external GFF3 file
prok_rm=0 #forces MAKER to repeatmask prokaryotes (no reason to change this), 1 = yes, 0 = no
softmask=1 #use soft-masking rather than hard-masking in BLAST (i.e. seg and dust filtering)
```

Gene Prediction

```
snaphmm= #SNAP HMM file
gmhmm= #GeneMark HMM file
augustus_species= #Augustus gene prediction species model
fgenesh_par_file= #FGENESH parameter file
pred_gff= #ab-initio predictions from an external GFF3 file
model_gff= #annotated gene models from an external GFF3 file (annotation pass-through)
est2genome=1 #infer gene predictions directly from ESTs, 1 = yes, 0 = no
protein2genome=0 #infer predictions from protein homology, 1 = yes, 0 = no
trna=0 #find tRNAs with tRNAscan, 1 = yes, 0 = no
snoscan_rrna= #rRNA file to have Snoscan find snoRNAs
unmask=0 #also run ab-initio prediction programs on unmasked sequence, 1 = yes, 0 = no
```

Other Annotation Feature Types

```
other_gff= #extra features to pass-through to final MAKER generated GFF3 file
```

External Application Behavior Options

```
alt_peptide=C #amino acid used to replace non-standard amino acids in BLAST databases
cpus=1 #max number of cpus to use in BLAST and RepeatMasker (not for MPI, leave 1 when using MPI)
```

MAKER Behavior Options

```
max_dna_len=100000 #length for dividing up contigs into chunks (increases/decreases memory usage)
min_contig=1 #skip genome contigs below this length (under 10kb are often useless)

pred_flank=200 #flank for extending evidence clusters sent to gene predictors
pred_stats=0 #report AED and QI statistics for all predictions as well as models
AED_threshold=1 #Maximum Annotation Edit Distance allowed (bound by 0 and 1)
min_protein=0 #require at least this many amino acids in predicted proteins
alt_splice=0 #Take extra steps to try and find alternative splicing, 1 = yes, 0 = no
always_complete=0 #extra steps to force start and stop codons, 1 = yes, 0 = no
map_forward=0 #map names and attributes forward from old GFF3 genes, 1 = yes, 0 = no
keep_preds=0 #Concordance threshold to add unsupported gene prediction (bound by 0 and 1)

split_hit=10000 #length for the splitting of hits (expected max intron size for evidence alignments)
single_exon=0 #consider single exon EST evidence when generating annotations, 1 = yes, 0 = no
single_length=250 #min length required for single exon ESTs if 'single_exon is enabled'
correct_est_fusion=0 #limits use of ESTs in annotation to avoid fusion genes

tries=2 #number of times to try a contig if there is a failure for some reason
clean_try=0 #remove all data from previous run before retrying, 1 = yes, 0 = no
clean_up=0 #removes theVoid directory with individual analysis files, 1 = yes, 0 = no
TMP= #specify a directory other than the system default temporary directory for temporary files
```

Basic MPI Example

Many of the steps used by MAKER can be computationally demanding, so MAKER will most commonly be run using MPI. However because of limited resources available for a large group this will be the only exercise where we actually run MAKER with MPI during the tutorial. We already covered briefly how to install MAKER with MPI support, and to load the currently installed MPI configuration for MAKER on the class servers you will need to load a couple of modules.

```
module load openmpi-1.10-x86_64
module load maker/2.31.9
```

Now let's move back to the first example directory.

```
cd ~/maker_tutorial/example_01_basic
ls -1
```

I've already placed the files you need in the directory.

```
dpp_contig.fasta
dpp_est.fasta
dpp_protein.fasta
finished1.tgz
finished2.tgz
hsap_contig.fasta
hsap_contig.maker.output
hsap_est.fasta
hsap_protein.fasta
opts1.txt
opts2.txt
```

We need to build new maker configuration files and populate the appropriate values.

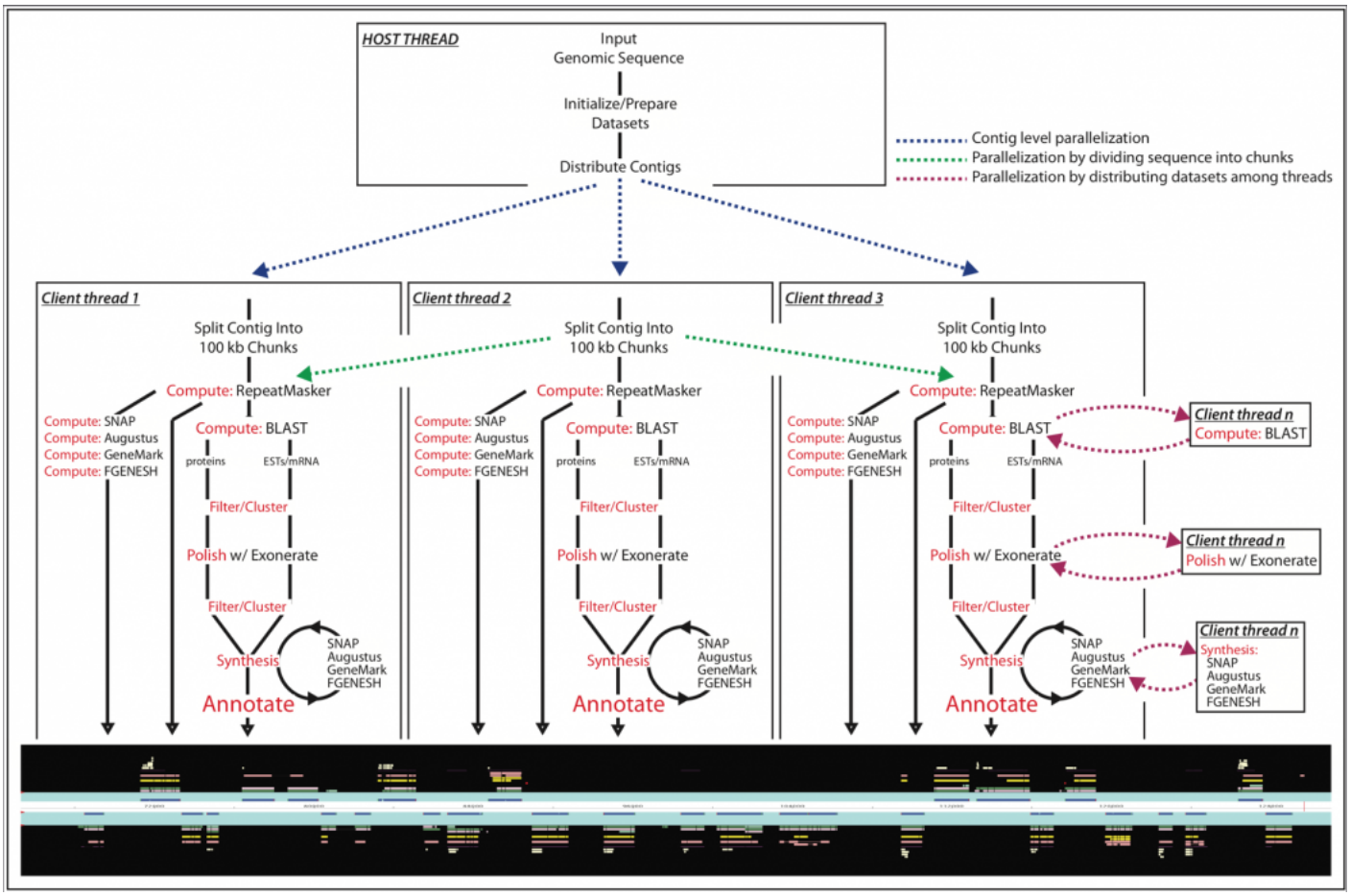
```
maker -CTL
nano maker_opts.ctl

  or

cp opts2.txt maker_opts.ctl
```

```
genome=hsap_contig.fasta
est=hsap_est.fasta
protein=hsap_protein.fasta
augustus_species=human
```

MAKER is now configured to generate annotations using a the gene predictor Augustus trained to predict on a human genome. We will discuss how training works for another algorithm in a later example. Just like our previous run will now launch MAKER, but this time we will configure it to run with MPI.

```
tar -zxf finished2.tgz
mpiexec -n 4 maker
```

Below you will see a workflow of how MAKER parallelizes steps under MPI.



MAKER can parallelize at various levels using MPI

Once the MPI run finishes you can load load the file `hsap_contig.maker.output/hsap_contig_datastore/80/99/NT_010783.15/NT_010783%2E15.gff` into JBrowse. Click below.

Click to see example 1.2 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example1.2/)

Training *ab initio* Gene Predictors

If you are involved in a genome project for an emerging model organism, you should already have an EST database, or more likely now mRANA-Seq data, which would have been generated as part of the original sequencing project. A protein database can be collected from closely related organism genome databases or by using the UniProt/SwissProt protein database or the NCBI NR protein database. However a trained *ab initio* gene predictor is a much more difficult thing to generate. Gene predictors require existing gene models on which to base prediction parameters. However, with emerging model organisms you are not likely to have any pre-existing gene models. So how then are you supposed to train your gene prediction programs?

MAKER gives the user the option to produce gene annotations directly from the EST evidence. You can then use these imperfect gene models to train gene predictor program. Once you have re-run MAKER with the newly trained gene predictor, you can use the second set of gene annotations to train the gene predictors yet again. This boot-strap process allows you to iteratively improve the performance of *ab initio* gene predictors.

I've created an example file set so you can learn to train the gene predictor SNAP using this procedure.

First let's move to the example directory.

```
cd ~/maker_tutorial/example_02_abinitio
ls -1
```

This directory looks a lot like the one from example_01. I've already placed the files you need in the directory.

```
finished1.tgz
finished2.tgz
finished3.tgz
opts1.txt
opts2.txt
opts3.txt
pyu_contig.fasta
pyu_est.fasta
sp_protein.fasta
```

We need to build maker configuration files and populate the appropriate values.

```
maker -CTL
nano maker_opts.ctl

  or

cp opts1.txt maker_opts.ctl
```

```
genome=pyu_contig.fasta
est=pyu_est.fasta
protein=sp_protein.fasta
est2genome=1
protein2genome=1
```

MAKER is now configured to generate annotations from the EST and aligned protein data, so start the program (this normally takes about 20 min to run because it's a long contig). But we are going to unpack the `partial.tgz` file before running maker to get some of our raw data precomputed and give it that nice "cooking show" feel (... or not it's your choice). We will use the -base command line flag to affect the output directory so we can run multiple ways and preserve output in separate directories (otherwise MAKER will overwrite to the same directory).

```
tar -zxf finished1.tgz
maker -base pyu_contig1
```

Once finished you can load load the file `pyu_contig.maker.output/pyu-contig_datastore/09/14/scf1117875582023/scf1117875582023.gff` into JBrowse. Click below.

Click to see example 2.1 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example2.1/)

You will see that there are far more regions with evidence alignments than there are gene annotations. This is because there are so few spliced ESTs and well aligned that are capable of generating gene models. But there are enough models to seed the training fro an ab initio gene predictor like SNAP.

To train SNAP, we need to convert the GFF3 gene models to ZFF format. To do this we need to collect all GFF3 files into a single directory.

```
mkdir snap
cd snap
gff3_merge -d ../pyu_contig1.maker.output/pyu_contig1_master_datastore_index.log
maker2zff pyu_contig1.all.gff
ls -1
```

There should now be two new files. The first is the ZFF format file and the second is a FASTA file the coordinates can be referenced against. These will be used to train SNAP.

```
genome.ann
genome.dna
```

The basic steps for training SNAP are first to filter the input gene models, then capture genomic sequence immediately surrounding each model locus, and finally uses those captured segments to produce the HMM. You can explore the internal SNAP documentation for more details if you wish.

```
fathom -categorize 1000 genome.ann genome.dna
fathom -export 1000 -plus uni.ann uni.dna
forge export.ann export.dna
hmm-assembler.pl pyu . > ../pyu1.hmm
cd ..
```

The final training parameters file is `pyu1.hmm`. We do not expect SNAP to perform that well with this training file because it is based on incomplete gene models; however, this file is a good starting point for further training.

We need to run MAKER again with the new HMM file we just built for SNAP.

```
nano maker_opts.ctl

  or

cp opts2.txt maker_opts.ctl
```

And set:

```
snaphmm=pyu1.hmm
est2genome=0
protein2genome=0
```

Now run again

```
tar -zxf finished2.tgz
mpiexec -n 4 maker -base pyu_contig2
```

Now let's look at the output in JBrowse.

Click to see example 2.2 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example2.2/)

When you examine the annotations you should notice that final MAKER gene models displayed in light blue, are more abundant now and are in relatively good agreement with the evidence alignments. However the SNAP *ab initio* gene predictions in the evidence tier do not yet match the evidence that well. This is because SNAP predictions are based solely on the mathematic descriptions in the HMM; whereas, MAKER models also use evidence alignments to help further inform gene models. This demonstrates why you get better performance by running *ab initio* gene predictors like SNAP inside of MAKER rather than producing gene models by themselves for emerging model organism genomes. The fact that the MAKER models are in better agreement with the evidence than the current SNAP models also means I can use the MAKER models to retrain SNAP in a bootstrap fashion, thereby improving SNAP's performance and consequentially MAKER's performance.

Let's retrain SNAP, and run MAKER again.

```
mkdir snap2
cd snap2
gff3_merge -d ../pyu_contig2.maker.output/pyu_contig2_master_datastore_index.log
maker2zff pyu_contig2.all.gff
fathom -categorize 1000 genome.ann genome.dna
fathom -export 1000 -plus uni.ann uni.dna
forge export.ann export.dna
hmm-assembler.pl pyu . > ../pyu2.hmm
cd ..
```

Change configuration file.

```
nano maker_opts.ctl

  or

cp opts3.txt maker_opts.ctl
```

```
snaphmm=pyu2.hmm
```

Run maker.

```
tar -zxf finished3.tgz
mpiexec -n 4  maker -base pyu_contig3
```

Let's examine the resulting GFF3 file one last time in JBrowse.

Click to see example 2.3 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example2.3/)

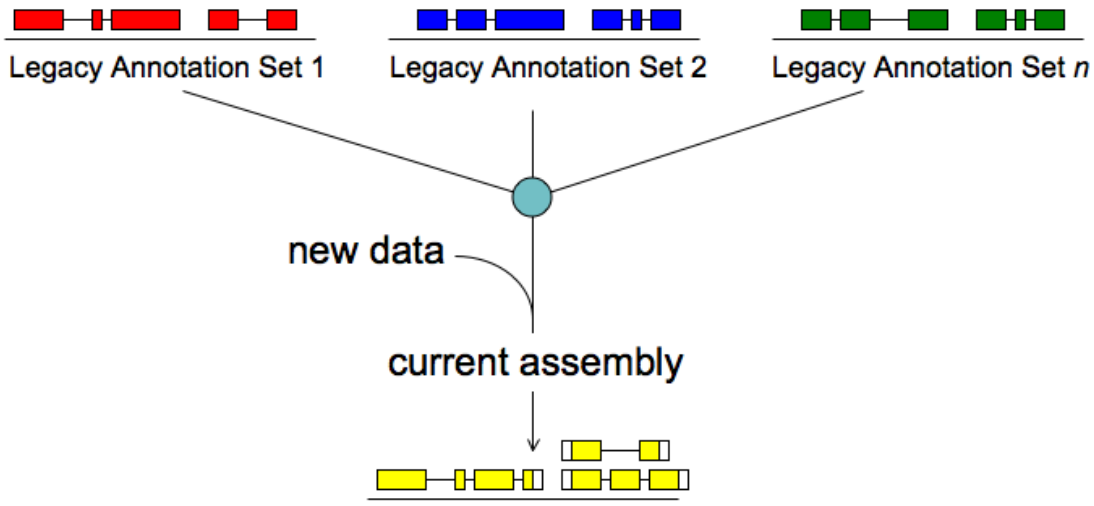
As you can see there, there is now a marked degree of improvement in both the MAKER and SNAP gene models, and both models are in more agreement with each other.

Merge/Resolve Legacy Annotations

Legacy annotations

- Many are no longer maintained by original creators

- In some cases more than one group has annotated the same genome, using very different procedures, even different assemblies
- Many investigators have their own genome-scale data and would like a private set of annotations that reflect these data
- There will be a need to revise, merge, evaluate, and verify legacy annotation sets in light of RNA-seq and other data



MAKER will:

- Identify legacy annotation most consistent with new data
- Automatically revise it in light of new data
- If no existing annotation, create new one

To make legacy annotation support possible MAKER uses a feature called GFF3 pass-through. Basically MAKER can take features from any source as long as you provide the data in GFF3 format. The following are GFF3 pass-through options. The most commonly used ones are pred_gff which is used to pass in ab initio gene predictions from programs without build in MAKER support and est_gff which can be used to add mRNA-seq data from programs like tophat and cufflinks (MAKER comes with the accessory scripts `tophat2gff3` and `cufflinks2gff3`).

- est_gff= #aligned ESTs or mRNA-seq from an external GFF3 file
- altest_gff= #aligned ESTs from a closly relate species in GFF3 format
- protein_gff= #aligned protein homology evidence from an external GFF3 file
- rm_gff= #pre-identified repeat elements from an external GFF3 file
- pred_gff= #ab-initio predictions from an external GFF3 file
- model_gff= #annotated gene models from an external GFF3 file (annotation pass-through)
- other_gff= #extra features to pass-through to final MAKER generated GFF3 file

We will be using the model_gff option to pass in legacy gene models.

So let's get started.

```
cd ~/maker_tutorial/example_03_legacy
ls -1

finished.tgz
```

```
legacy_contig.fasta
legacy_set1.gff
legacy_set2.gff
opts.txt
pyu_est.fasta
pyu.hmm
sp_protein.fasta
```

Now create maker control files and either edit the values or copy the `opts.txt` file.

```
maker -CTL
nano maker_opts.ctl

   or

cp opts.txt maker_opts.ctl
```

```
genome=legacy_contig.fasta
est=pyu_est.fasta
protein=sp_protein.fasta
model_gff=legacy_set1.gff,legacy_set2.gff
snapphmm=pyu.hmm
```

Notice the comma separated lists used by model_gff=. That is a MAKER feature available to all of the input options that take files as their value. As an additional feature you can also label the output with tags that can provide more context using ':' for separation. For example instead of `est=pyu_est.fasta`, I could put `est=pyu_est.fasta:hypoxia` for ESTs collected from a low oxygen study. Then this tag will be added to the GFF3 file, and instead of `est2genome` in the source column you would get `est2genome:hypoxia`. Many GMOD programs know how to use these extra tags, for example GBrowse can provide context specific "on/off" boxes and grouping for proteins, ESTs, and other evidence types based on their values.

Try a label if you want.

```
nano maker_opts.ctl

est=pyu_est.fasta:hypoxia
```

Now let's run MAKER.

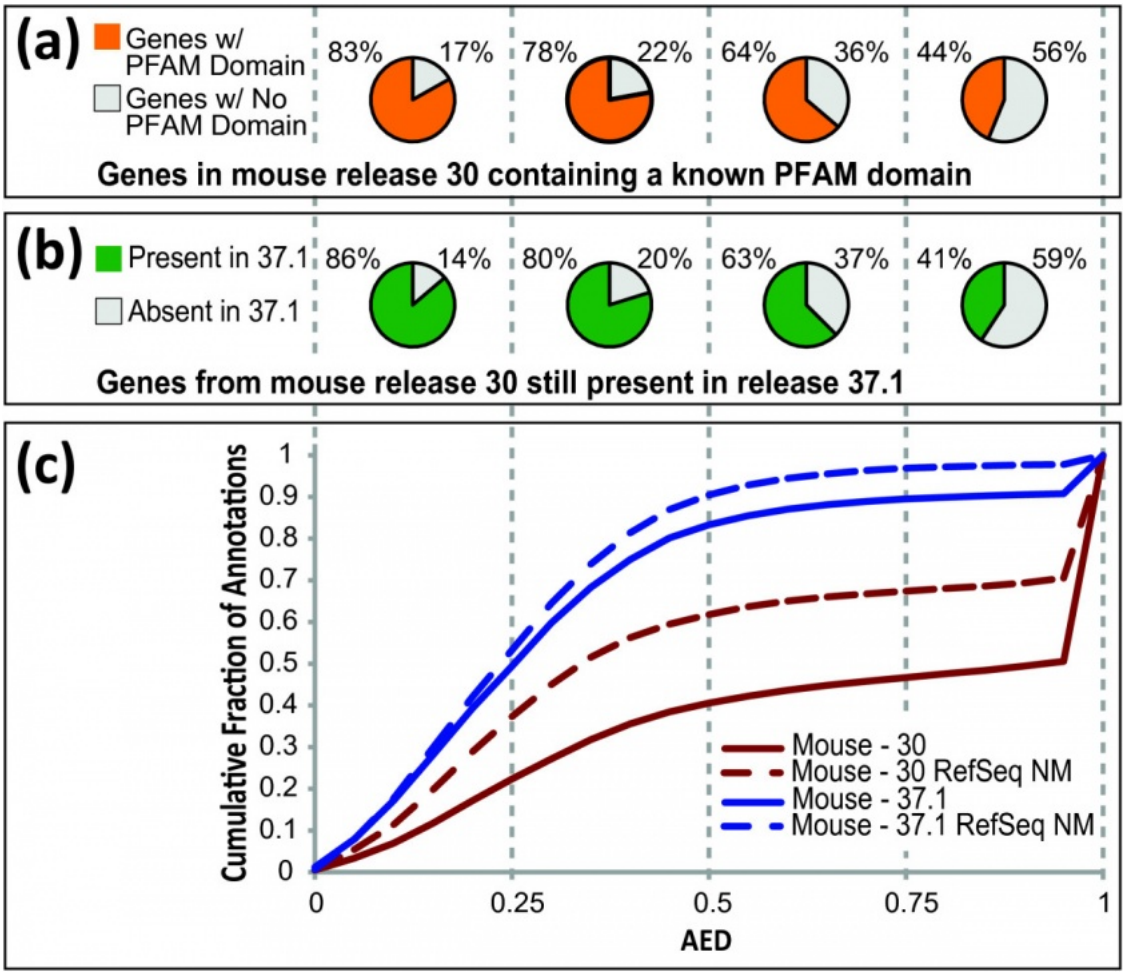
```
tar -zxf finished.tgz
mpiexec -n 4 maker
```

When MAKER finishes you can review the output in JBrowse.

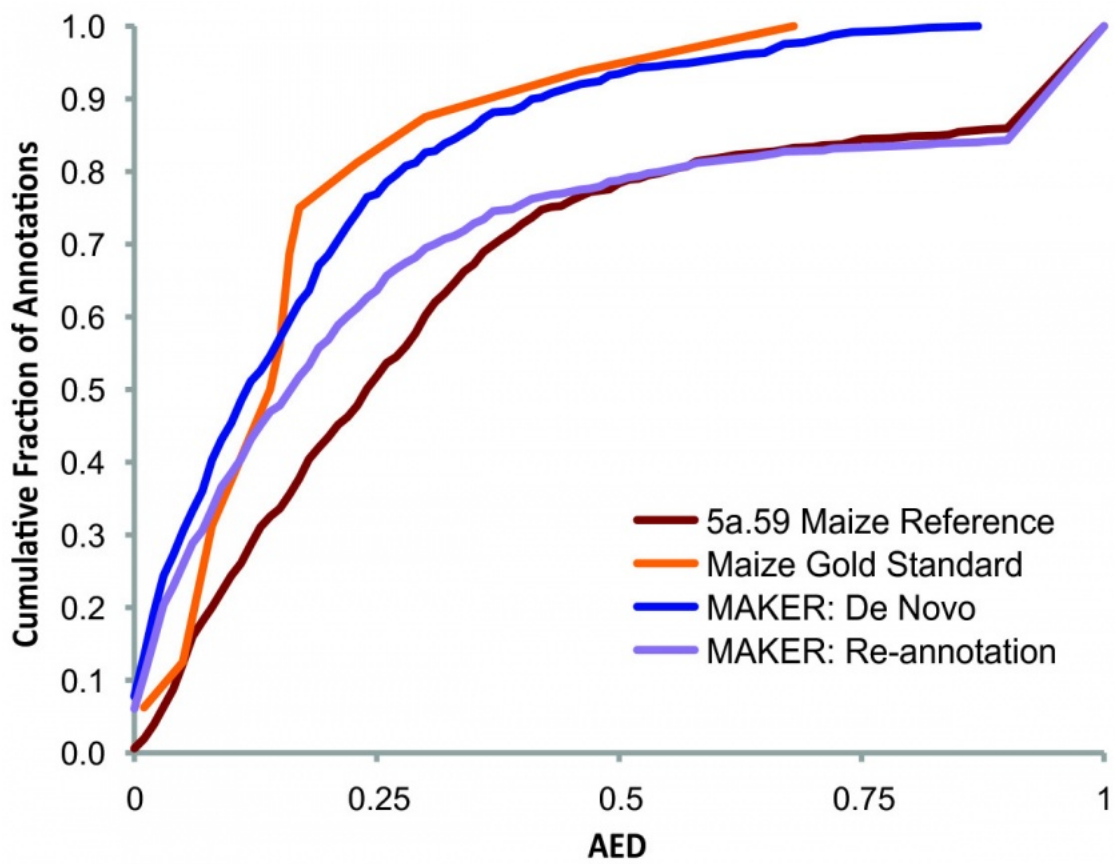
Click to see example 3 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example3/)

You will see that some of the new annotations come from legacy_set_1, some from legacy_set_2, and some are new models produced by SNAP. How they are chosen is based on how well they match the evidence which is measured using the metric AED.

Improving Annotation Quality with MAKER's AED score



Re-annotation with MAKER



Re-annotation and update of Maize with MAKER

Post Processing of Annotations

Once you've determined where the genes are the next question is what do they do. Also it would be nice to change ugly MAKER assigned gene names to follow more standardized formats.

MAKER has a number of accessory scripts that allow you to do just that. So let's take a look at our last example.

```
cd ~/maker_tutorial/example_04_postannotation
ls -1

 hsap_contig.gff
 hsap_contig.maker.proteins.fasta
 hsap_contig.maker.transcripts.fasta
 output.blastp
 output.iprscan
 uniprot_sprot.db
```

Here we have our MAKER output GFF3 and FASTA files for proteins and transcripts (Click to see GFF3 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example4.1/)). We also have output reports from the program InterProScan (https://www.ebi.ac.uk/interpro/about.html) (a domain finder) ran on the MAKER proteins and a BLAST report of homology of the MAKER proteins to UniProt/Swiss-Prot. How to run these programs is not part of this tutorial, but how to integrate their output is.

This is an example command line for running BLASTP against UniProt/Swiss-Prot (you don't need to run it, it's just for reference):

```
blastp -query hsap_contig.maker.proteins.fasta -db uniprot_sprot.fasta -evalue 1e-6 -max_hsps 1 -max_target_seqs 1 -outfmt
6 -out output.blastp
```

This is an example command line for running InterProScan (you don't need to run it, it's just for reference):

```
interproscan.sh -appl pfam -dp -f TSV -goterms -iprlookup -pa -t p -i hsap_contig.maker.proteins.fasta -o output.iprscan
```

But first lets fix those ugly MAKER names.

MAKER comes with the script maker_map_ids to make it easy to rename genes and follow formats such as those suggested by NCBI (organism prefix and gene numbers).

```
Synopsis:

maker_map_ids --prefix PYU1_ --justify 8 genome.all.gff > genome.all.id.map

Description:

This script wil take a GFF3 file and create a mapping file of gene and
transcript IDs to more numerically incremented human friendly unique
IDs.

Options:

  --prefix      The prefix to use for all IDs (default = 'MAKER_')
  --suffix      A suffix to use for all transcript IDs (default = '-R').
                Specifying --suffix will also turn on --iterate.
  --initial     The initial value to start with for ID count. If --initial
                is supplied more than once than the first value will be used
                for genes and the second for transcripts (default = 1)
  --abrv_gene   Optional abreviation added to IDs for genes (i.e. = 'G')
  --abrv_tran   Optioanl abreviation added to IDs for transcripts (i.e. = 'T')
  --iterate     Transcript iteration to add to IDs. Value can be '0', '1',
                or 'A', i.e. mRNA-0 or mRNA-1 or mRNA-A (default = 'A')
  --justify     The unique integer portion of the ID will be right justified
                with '0's to this length (default = 8)
  --sort_order  A tab delimited file containing two columns: contig_id
                and sort_order.  Genes and transcripts will be named
                in consecutive order along the contigs, and the
                contigs will be sorted in the order specified by the
                file.  If sort_order is not given and there are
                ##sequence-region directives at the top of the gff
                file then naming will be ordered by decreasing contig
                length.

Formating:
  By default gene and transcript abreviations will appear at the end of the
  suffix and the iterate value will appear at the end of the prefix (only on
  transcripts).  You can specify an alternate location for the abbreviation by
  placing a '?' character as part of the prefix or the sufix. An alternate
  location for the iterator can be given by adding a % character to the
  suffix or prefix.
```

We will use the prefix 'GMOD' for our gene names, and an eight digit identifier.

```
maker_map_ids --prefix GMOD_ --justify 8 hsap_contig.gff > hsap_contig.map
```

The output is a two column file translating old gene and mRNA names to new more standardized names.

```
less hsap_contig.map

 maker-NT_010783.15-snap-gene-0.0       GMOD_00000001
 maker-NT_010783.15-snap-gene-0.0-mRNA-1        GMOD_00000001-RA
```

```
maker-NT_010783.15-snap-gene-0.1       GMOD_00000002
maker-NT_010783.15-snap-gene-0.1-mRNA-1        GMOD_00000002-RA
...
```

We now need to apply the new name to any files containing the old names. The following scripts are used for that.

- map_fasta_ids - fixes names in FASTA files
- map_gff_ids - fixes names in GFF3 files and added Alias= attributes that allow recovery of old names
- map_data_ids - tries to fixes names in any generic data file

These script do in-place replacement of names, so lets copy the files before running the scripts.

```
cp hsap_contig.gff hsap_contig.renamed.gff
cp hsap_contig.maker.proteins.fasta hsap_contig.maker.proteins.renamed.fasta
cp hsap_contig.maker.transcripts.fasta hsap_contig.maker.transcripts.renamed.fasta
cp output.iprscan output.renamed.iprscan
cp output.blastp output.renamed.blastp
map_gff_ids hsap_contig.map hsap_contig.renamed.gff
map_fasta_ids hsap_contig.map hsap_contig.maker.proteins.renamed.fasta
map_fasta_ids hsap_contig.map hsap_contig.maker.transcripts.renamed.fasta
map_data_ids hsap_contig.map output.renamed.iprscan
map_data_ids hsap_contig.map output.renamed.blastp
```

You can see names have changed by looking at the files.

```
less -S hsap_contig.renamed.gff
```

Next lets add putative gene functions using output in a BLAST job of our proteins against UniProt/Swiss-Prot. Here we use the following scripts:

- maker_functional_gff - adds putative functions from BLAST report to GFF3 files (supports UniProt/Swiss-Prot headers)
- maker_functional_fasta - adds putative functions from BLAST report to FASTA files (supports UniProt/Swiss-Prot headers).

This once again is an example command line for running BLASTP against UniProt/Swiss-Prot:

```
blastp -query hsap_contig.maker.proteins.fasta -db uniprot_sprot.fasta -evalue 1e-6 -max_hsps 1 -max_target_seqs 1 -outfmt
6 -out output.blastp
```

Use these commands to update your annotations with information from the BLAST report:

```
maker_functional_gff uniprot_sprot.db output.renamed.blastp hsap_contig.renamed.gff >
hsap_contig.renamed.putative_function.gff
maker_functional_fasta uniprot_sprot.db output.renamed.blastp hsap_contig.maker.proteins.renamed.fasta >
hsap_contig.maker.proteins.renamed.putative_function.fasta
maker_functional_fasta uniprot_sprot.db output.renamed.blastp hsap_contig.maker.transcripts.renamed.fasta >
hsap_contig.maker.transcripts.renamed.putative_function.fasta
```

Look at the files to see that putative functions were added.

```
less -S hsap_contig.maker.proteins.renamed.putative_function.fasta
```

Finally we will add protein domain information to the final annotations using a report from InterProScan. This is done using the following scripts:

- ipr_update_gff - adds searchable tags to the gene and mRNA features in the GFF3 files
- iprscan2gff3 - adds physical viewable features for daomains that can be displayed in JBrowse, Gbrowse, and Web Apollo.

This once again is an example command line for running InterProScan:

```
interproscan.sh -appl pfam -dp -f TSV -goterms -iprlookup -pa -t p -i hsap_contig.maker.proteins.fasta -o output.iprscan
```

Use these commands to update your annotations with information from the InterProScan report:

```
ipr_update_gff hsap_contig.renamed.putative_function.gff output.renamed.iprscan >
hsap_contig.renamed.putative_function.domain_added.gff
iprscan2gff3 output.renamed.iprscan hsap_contig.renamed.gff > visible_iprscan_domains.gff
```

Now look at the original annotations in JBrowse and compare it to the final annotations, to see how adding new names, domains, and putative functions can greatly improve the utility of your genome database.

Click to see example 4.1 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example4.1/)

Click to see example 4.2 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example4.2/)

MAKER Accessory Scripts

MAKER comes with a number of accessory scripts that assist in manipulations of the MAKER input and output files.

Scripts:

- *gff3_merge* - Collects all of MAKER's GFF3 file output for each contig and merges them to make a single genome level GFF3

  ```
  gff3_merge -d <datastore_index> -o <outfile>
  ```

- *fasta_merge* - Collects all of MAKER's fasta file output for each contig and merges them to make genome level fastas

  ```
  fasta_merge -d <datastore_index> -o <outfile>
  ```

- *maker_map_ids* - Build shorter IDs/Names for MAKER genes and transcripts following the NCBI suggested naming format.

  ```
  maker_map_ids --prefix PYU1_ --justify 6 genome.all.gff > genome.all.id.map
  ```

- *map_gff_ids* - Maps short IDs/Names to MAKER GFF3 files, old IDs/Names are mapped to to the Alias attribute.

  ```
  map_gff_ids <map_file> <gff3_file>
  ```

- *map_fasta_ids* - Maps short IDs/Names to MAKER fasta files.

```
map_fasta_ids <map_file> <fasta_file>
```

- *map_data_ids* - This script takes a id map file and changes the name of the ID in a data file. The map file is a two column tab delimited file with two columns: old_name and new_name. The data file is assumed to be tab delimited by default, but this can be altered with the delimit option. The ID in the data file can be in any column and is specified by the col option which defaults to the first column.

```
map_data_ids genome.all.id.map data.txt
```

- *maker_functional_fasta* - Maps putative functions identified from BLASTP against UniProt/SwissProt to the MAKER produced tarnscript and protein fasta files.

```
maker_functional_fasta <uniprot_fasta> <blast_output> <fasta1> <fasta2> <fasta3> ...
```

- *maker_functional_gff* - Maps putative functions identified from BLASTP against UniProt/SwissProt to the MAKER produced GFF3 files in the Note attribute.

```
maker_functional_gff <uniprot_fasta> <blast_output> <gff3_1>
```

- *iprscan2gff3* - Takes InerproScan (iprscan) output and generates GFF3 features representing domains. Interesting tier for GBrowse.

```
iprscan2gff3 <iprscan_file> <gff3_fasta>
```

- *ipr_update_gff* - Takes InterproScan (iptrscan) output and maps domain IDs and GO terms to the Dbxref and Ontology_term attributes in the GFF3 file.

```
ipr_update_gff <gff3_file> <iprscan_file>
```

- *fasta_tool* - The script can search, reformat, and manipulate a fasta file in a variety of ways.

```
fasta_tool [OPTIONS] <fasta_file>
```

- *maker2zff* - Pulls out MAKER gene models from the MAKER GFF3 output and convert them into ZFF format for SNAP training.

```
maker2zff <gff3_file>
```

- *maker2eval_gtf* - This script converts MAKER GFF3 files into GTF formated files for the program EVAL (an annotation sensitivity/specificity evaluating program). The script will only extract features explicitly declared in the GFF3 file, and will skip implicit features (i.e. UTR, start codons, and stop codons). To extract implicit features to the GTF file, you will first need to expicitly declare them in the GFF3 file. This can be done by calling the script add_utr_to_gff3 to add formal declaration lines to the GFF3 file.

```
maker2eval_gtf <maker_gff3_file>
```

- *maker2jbrowse* - This script will produce a JBrowse data set from MAKER gff3 files.

```
maker2chado [OPTION] <database_name> <gff3file1> <gff3file2> ...
```

- *maker2chado* - This script takes MAKER produced GFF3 files and dumps them into a Chado database. You must set the database up first according to CHADO installation instructions. CHADO provides its own methods for loading GFF3, but this script makes it easier for MAKER specific data. You can either provide the datastore index file produced by MAKER to the script or add the GFF3 files as command line arguments.

```
maker2chado [OPTION] <database_name> <gff3file1> <gff3file2> ...
```

- *chado2gff3* - This script takes default CHADO database content and produces GFF3 files for each contig/chromosome.

```
chado2gff3 [OPTION] <database_name>
```

Example MAKER Annotation Project

You should now have the sufficient understanding of how to use MAKER to perform your own small annotation project. For this example we will do just that using an assembly of *Schizosaccharomyces pombe* chromosome III.

The basic steps to any annotation project are as follows:

1. Collect homology evidence datasets (we will provide those for this example)
2. Build RepeatMasker library (RepBase fortunately already has sufficient *S. pombe* repeats annotated for us to just use that)
3. Train *ab initio* gene predictors
4. Computational annotation (run MAKER)
5. Review/curate annotations
6. Add functional annotations and meta-data
7. Distribute results

Now let's move to the example directory.

```
cd ~/maker_tutorial/example_05_yeast
ls -1

 finished1.tgz
 finished2.tgz
 finished3.tgz
 finished4.tgz
 opts1.txt
 opts2.txt
 S_pombe_chrIII.fa
 S_pombe_trinity_assembly.fasta
 Swissprot_no_S_pombe.fasta
```

The example directory already contains datasets as well as preconfigured control files and partial results to assist with runtimes. You should try and complete this part based on your own decisions, but I will detail steps below to assist incase you become lost at any point.

Collect homology evidence datasets

You should provide both transcript and protein homology evidence. Transcript evidence should be from the organism being annotated and is generally sequenced simultaneously with the genome and prepared with tools such as Trinity (https://github.com/trinityrnaseq/trinityrnaseq/wiki). You can also collect sequencing data from resources such as NCBI's Short Read Archive (https://www.ncbi.nlm.nih.gov/sra). For protein datasets, you can provide proteomes from at least two related organisms and a curated datatset such as UniProt/Swiss-Prot (http://www.uniprot.org). You can provide additional proteomes and even an outgroup to maximize sensitivity.

Build RepeatMasker library

An external repeat library can be prepared using tools such as RepeatModeler (http://www.repeatmasker.org). It is recommended to provide the species specific library together with RepBase (http://www.girinst.org/repbase/).

Train *ab initio* gene predictors

Below are suggested options for training SNAP.

```
genome=S_pombe_chrIII.fa
est=S_pombe_trinity_assembly.fasta
protein=Swissprot_no_S_pombe.fasta
est2genome=1
protein2genome=1
```

Click to see example 5.1 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example5.1/)

SNAP training protocol.

```
mkdir snap
cd snap
gff3_merge -d ../S_pombe_chrIII.maker.output/S_pombe_chrIII_master_datastore_index.log
maker2zff S_pombe_chrIII.all.gff
fathom -categorize 1000 genome.ann genome.dna
fathom -export 1000 -plus uni.ann uni.dna
forge export.ann export.dna
hmm-assembler.pl spombe . > ../spombe1.hmm
cd ..
```

Click to see example 5.2 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example5.2/)

More than one round of training is not always necessary for fungi, as they tend to have simpler intron/exon structures.

Computational annotation

Using more than one gene predictor is recommended when possible. It provides additional alternatives for evaluations using the AED calculation.

```
genome=S_pombe_chrIII.fa
est=S_pombe_trinity_assembly.fasta
protein=Swissprot_no_S_pombe.fasta
est2genome=0
protein2genome=0
snaphmm=spombe1.hmm
augustus_species=saccharomyces
always_complete=1
```

```
single_exon=1
correct_est_fusion=1
```

Click to see example 5.3 in JBrowse (http://weatherby.genetics.utah.edu/maker_tutorial/example5.3/)

Review/curate annotations

I'll leave this for the Web Apollo section, but other tools for annotation improvement include Evidence Modeler (https://evidencemodeler.github.io) (better annotations for organisms with complex splicing evidence) and Defusion (https://github.com/wjidea/defusion) (fixes false gene merges causes by evidence that bridges across neighboring paralogs and falsely merged mRNA-seq assemblies).

Add functional annotations and meta-data

Perform external functional analysis.

```
blastp -query S_pombe_chrIII.all.maker.proteins.fasta -db Swissprot_no_S_pombe.fasta -evalue 1e-6 -max_hsps 1 -
max_target_seqs 1 -outfmt 6 -out S_pombe_chrIII.all.maker.proteins.fasta.blastp
interproscan.sh -appl pfam -dp -f TSV -goterms -iprlookup -pa -t p -i S_pombe_chrIII.all.maker.proteins.fasta -o
S_pombe_chrIII.all.maker.proteins.fasta.iprscan
```

Rename gene models and other data.

```
maker_map_ids --prefix GMOD_ --justify 8 S_pombe_chrIII.all.gff > map
map_gff_ids map S_pombe_chrIII.all.gff
map_fasta_ids map S_pombe_chrIII.all.maker.proteins.fasta
map_fasta_ids map S_pombe_chrIII.all.maker.transcripts.fasta
map_data_ids map S_pombe_chrIII.all.maker.proteins.fasta.blastp
map_data_ids map S_pombe_chrIII.all.maker.proteins.fasta.iprscan
```

Integrate functional annotations into structural annotations.

```
ipr_update_gff S_pombe_chrIII.all.gff S_pombe_chrIII.all.maker.proteins.fasta.iprscan > S_pombe_chrIII.all.2.gff
iprscan2gff3 S_pombe_chrIII.all.maker.proteins.fasta.iprscan S_pombe_chrIII.all.gff >> S_pombe_chrIII.all.2.gff
mv S_pombe_chrIII.all.2.gff S_pombe_chrIII.all.gff

maker_functional_gff Swissprot_no_S_pombe.fasta S_pombe_chrIII.all.maker.proteins.fasta.blastp S_pombe_chrIII.all.gff >
S_pombe_chrIII.all.2.gff
maker_functional_fasta Swissprot_no_S_pombe.fasta S_pombe_chrIII.all.maker.proteins.fasta.blastp
S_pombe_chrIII.all.maker.proteins.fasta > S_pombe_chrIII.all.maker.proteins.2.fasta
maker_functional_fasta Swissprot_no_S_pombe.fasta S_pombe_chrIII.all.maker.proteins.fasta.blastp
S_pombe_chrIII.all.maker.transcripts.fasta > S_pombe_chrIII.all.maker.proteins.2.fasta
mv S_pombe_chrIII.all.2.gff S_pombe_chrIII.all.gff
mv S_pombe_chrIII.all.maker.proteins.2.fasta S_pombe_chrIII.all.maker.proteins.fasta
mv S_pombe_chrIII.all.maker.proteins.fasta S_pombe_chrIII.all.maker.proteins.2.fasta
```

Distribute results

This is where you branch out to other GMOD tools, such as JBrowse (https://jbrowse.org), Chado (http://gmod.org/wiki/Chado), and Tripal (http://tripal.info).
