

The est2assembly manual

by Alexie Papanicolaou, MPI for Chemical Ecology & University of Exeter
Homepage: <http://www.rfc.ex.ac.uk/software>

Preface

This is the manual for the est2assembly pipeline, created to process Sanger/454 EST projects and produce a GBrowse driven interface. It aims to standardize the methods used in EST processing and being modular you can opt to use part of the pipeline.

The platform has three main modules: i) est_process handles the production of input files for the assemblers; ii) parameterize_assembly produces a series of different assemblies and by calculating quality statistics allows users to pick the most desirable assembly; iii) and analyze, a collection of Perl programs aimed at annotating, analyzing and distributing the assembly via GFF files which can be processed by BioPerl for databasing.

This manual is written with two audiences in mind: novices in Unix and seasoned travellers, apologies, therefore, if some things are put too simply. If something is not clear, however, do let me know and I'll update the documentation.

This is academic software so it will be rough around the edges, but if (or better: when) something does not work, please do let me know and I'll fix it as soon as possible.

Likewise, being open-source, if anyone wishes to contribute, please let me know and we can work together for the enhanced next version.

Contents

[Installation Novice](#)

[Installation Expert](#)

[Manual conventions](#)

[Downloading mRNA from GenBank](#)

[Running a BaseCall for Sanger capillary sequencing](#)

[Preprocessing your data](#)

[Running an Sanger and/or 454 assembly with parameterization](#)

[Assembly annotation](#)

[Creating GFF files](#)

[Populating a SeqFeature database for GBrowse](#)

[Interrogating the dataset and understanding the structure](#)

[Using a LSF-driven PC-Farm](#)

[est2assembly citation](#)

#v0.034 04Jul09 AP

Installation Novice

Dear novice user,

This platform can be used even by novice users, as long as they know basic Unix. A decent tutorial by Lincoln Stein is available here: http://stein.cshl.org/genome_informatics/

Once you are familiar with the basics, you can proceed. If you have any questions at any time, please email me alexie -alpha symbol- butterflybase.org

Please do follow the instructions below in order to have est2assembly running smoothly. I'll now walk you through it. In particular, we are going to install the Perl libraries (modules) which est2assembly relies on. CPAN is the repository where this libraries can be stored and the program cpan can be used to download, test and install them.

For the rest of this manual, where a \$ dollar symbol appears, it will represent your shell where you can type commands. It is accessed through the terminal application and comes in various flavours: bash, sh etc. (if you have a choice, I recommend bash as the most user-friendly and least troublesome). In any case, the one already installed will do fine.

Where a > sign appears, then it is commands that follow the previous ones after you press enter (e.g. the cpan shell)

For most of the installation, root (administrator) access is not required but recommended.

Depending on your linux distribution, CPAN is probably installed by default but likely to be severely out of date. Try typing:

```
$ cpan
```

If that does not work, try installing it. In a debian based system (such as the very user friendly Ubuntu) run this as root:

```
$ apt-get install cpan
```

or using sudo (if you are allowed, invoke root privileges without logging in)

```
$ sudo apt-get install cpan
```

If it is installed, then it is highly recommended that you (or the admin) update it to the current version using administrative privileges. Keeping CPAN up-to-date is something recommended for any perl program not just this platform.

```
$ sudo cpan
```

```
> install Bundle::CPAN
```

```
> exit  
> sudo cpan
```

This will take a long time and after re-invoking cpan you will be asked a lot of questions. I hope they are self-explanatory (defaults usually do) or you can figure it out by reading the cpan manual/website as it is beyond the scope of this walk-through. Always ask someone who knows if you are in doubt.

Now that cpan is up-to-date we can continue. We will install the libraries used by est2assembly, then the development branch of BioPerl (as currently the stable one is just too old) and finally invoke the GBrowse installation.

This script will do the all the steps in order. First read the manual (as you should do with any script first!)

```
$ perl doc install_perl_modules.pl
```

Then once you understand it, run it as root (recommended) or with the -local installation (if you cannot)

```
$ ./install_perl_modules.pl <options here>
```

The GBrowse installation can be tricky but normally it runs without problems. I have to refer you to their website if you need help: <http://gmod.org>

Please note that you don't need to have GBrowse installed to run this pipeline. But you do need the development version of BioPerl.

The next (and last) program which must be installed is the emboss package. EMBOSS is a very important package in bioinformatics and if you haven't installed it yet, I hope you find it useful beyond this pipeline.

Installation must be done as an administrator or with admin privileges (sudo). In Ubuntu (or other debian) simply

```
$ sudo apt-get emboss
```

Other systems, please refer to the website (<http://emboss.org>)

Last is the convenient but not necessary step of adding the path of est2assembly to your "executable path" (the \$PATH environmental variable) or copying all the perl scripts to your bin.

To put the directories in the path you can execute the following command from the directory where this file resides.

```
$ export PATH=$PATH:$PWD/preprocess:$PWD/
```

The \$PATH is your old PATH variable which is propagated, the \$PWD is a special variable which contains your current directory. Depending on your shell, if you press tab it will autocomplete to the current directory.

try

```
$ echo $PWD
```

and you will get the real full path of your current dir.

If you want this information to be stored when you start your computer, first get the line with \$PWD replaced to the real full path.

Then in your startup file for your shell (for bash it's [~/.bashrc](#)) where ~ is a special character/shortcut for your home. To install it for all users you can put it at [/etc/bash.bashrc](#)

e.g.

```
$ pico ~/.bashrc
```

or

```
$ vi ~/.bashrc
```

Alternatively, if you already have a [~/bin/](#) directory and it's in your path, you can run

```
$ install_into_bin.sh
```

To copy the files to your [~/bin/](#) (or other directory: remember to read the manual with perldoc).

Finally you have to install some external programs. Please read the INSTALL file for the relevant websites. We include those we are allowed by their license to.

That's it! For expert users this is probably trivial and will not take more than a few seconds but I hope this short manual is sufficient in detail for new administrators.

If you think it can be improved, please do send suggestions to alexie -alpha symbol- butterflybase.org. Even better, you can correct/expand it and email it and I'll include it in the next release (along with a big thank you).

Installation Expert

Dear expert user,

I hope the platform is self-explanatory to you, especially after reading this manual. The perl libraries can be installed by reading the Cpan_packages_needed.pm file to see which are needed, then also install the -dev branch of BioPerl and if you wish GBrowse.

Or you can do all of the above using the install_perl_modules.pl script.

You can put the scripts in your path or copy them to your bin. Please do keep the lib TrimByWindow properly connected or install it site-wise. Also the .config files are not needed as they can be specified by the command line interface but if you do want to avoid typing, please put them in the same

directory as the relevant scripts (\$RealBin/ is used to find them).

That's it! If you think it the manual can be improved, please do send suggestions to alexie -alpha symbol- butterflybase.org. Even better, you can correct/expand it and email it and I'll include it in the next release (along with a big thank you).

Manual conventions

For the rest of the manual, I would like to point out to the following conventions:

- Dollar symbol \$ denotes the command line.
- The brackets <> symbolise an option which can be replaced with text of your choosing. Don't include <> in the command line unless otherwise specified.
- The perldoc command (\$ perldoc <perlscript>) can be used to see the documentation of any script. Limited usage information can be gained by running the script without any options.
- An option can be activated with the option's full name or an abbreviation that is unique versus all the other available options in the script. Alternatives are shown separated by the horizontal bar | so that if two options/switches, d|debug and dir, exist then using -d will call for -debug and -di will call for -dir.
- The options sometimes have :i, :f or :s. :i denotes that an integer is expected for this option, :f expects a float (i.e. decimal) and :s it expects a string, i.e. a text. If nothing is given, then the option does not accept values but is a switch.
- If you include spaces in your values, then you must quote the value (e.g. in " "). Also if an option has the {,} denotation, it means that more than one value can be given to this option, either through specifying multiple times the switch or putting the values one after the other. See \$ perldoc Getopt::Long for detailed information for this system.

e.g.

```
$ ic_loaddata.pl
```

Please provide caf/p4e/annotation files or a config directory

Usage:

'c caffiles:s{0,}'	=> CAF assembly files
'p p4efiles:s{0,}'	=> prot4EST files.
'b blast annot8rfiles:s{0,}'	=> includes blasts, snps, annot8r, ipr etc
'host:s'	=> DBHOST
'port:i'	=> DBPORT
'prefix:s'	=> Prefix for psql database (def "ic_")
'create d drop recreate'	=> Drop and recreate psql database,
'a config gbrowse_config:s'	=> directory to gbrowse.conf which has include files
'species:s{,}'	=> Species list
'types:s{,}'	=> caf, orf or p4e

```
'debug'
```

```
=> Don't delete load tmp files
```

So the -caffiles can be given as -c and the -create can be given as -cr or as -d. Caffiles, p4efiles and annot8rfiles can have multiple values passed to them (with a minimum of 0, i.e. none). The -debug and -create options don't expect any values and if one is given, it will be ignored.

Downloading mRNA from GenBank

I have included a simple script to download EST, mRNA or even protein sequences from EBI (European Bioinformatics Institute), part of the GenBank consortium.

```
$ srsdownload.pl -s "Drosophila melanogaster" -m cDNA
```

will get all the cDNA (mRNA+EST) from D.mel. It will take some time as the script waits every 1000 requests so that their server doesn't complain.

In a transcriptome assembly we include these sequences (as Sanger technology) in order to improve the quality of the assembly (especially full length cDNA). MIRA allows the use of strain information so you can use

```
$ mira_strain.pl
```

to produce a strain file for GenBank data to concatenate into the final strain file.

Running a BaseCall for Sanger capillary sequencing

The script which drives the basecalling for sanger capillary sequencing is called (surprisingly) process_sanger_trace.pl.

As always, read the manual...

```
$ perldoc process_sanger_trace.pl
```

You can find where it is installed (if you didn't install it yourself)

```
$ which process_sanger_trace.pl
```

and maybe using

```
$ ls -l <output of above>
```

To see if it is a symbolic link to another directory. The configuration file should be in the same directory as the script.

We find that the newest (beta) version of phred is quite good (as good or better than the KB Basecaller if you use an ABI sequencer) but which basecaller you wish to use, is up to you. Note that the *Trimbywindow* routine can only be used with sequences which have integrated the KB basecalling in the .abi file.

The configuration file, residing in the same directory as the script, can be configured to recognise your sequence naming scheme (hopefully you used a scheme and not ad-hoc named each sequence...). You will do know some basics about Perl regular expressions (regex). E.g. see <http://www.cs.tut.fi/~jkorpela/perl/regexp.html>

Preprocessing your data

It is essential to preprocess your sequence data before trying to assemble it. You don't want to allow false local alignments in the assembly as these will not only result in misassemblies but also a high percentage of read rejection.

You probably want to mask/remove adaptor sequences added during the library construction, identify any vector sequence (if sequencing with capillaries) and common contaminants. Identifying and removing known repeats and transposons is also highly recommended as it will result in having 'cleaner' data for the assembler to process. If you are interested in transposons, then by having the above identification, you can annotate them separately.

The script `preprocessest.pl` (in the folder `est_assembly`) does all this for you in one go. It can be slow but it is thorough. Most importantly, it keeps a log file to what happened in each sequence so we can check (a database version is in my "to-do" list) As always, read the manual

```
$ perl doc preprocessest.pl
```

- It has a config file to specify database variables if you are using it often. But you can override them from the command line. If you don't want to use a specific database run at all (e.g. there is no adaptor sequence in your data (?!)) then please give the relevant -no option (e.g. -noadaptor).
- You can choose to keep the intermediate files, archive them in a tar.bz2 file or delete them. They do take an awful lot of space but they can give you valuable information about your data.
- The files useful to the assembler are the ones which have the word cleaned in them. They have the full sequence masked with Xs ("just in case" TM) and an XML files tells the MIRA assembler which parts to use. If you want, you can use the original sequence to give to the assembler (if

you think there is over-masking and wish to take advantage of MIRA's option to identify such regions). Use this command

```
$ trim_fasta_all.pl -inv -id <yourcleaned file> <your original file>
```

This will produce two files .trim and .discard. The .trim you wish to use, the discard you can look at and see what preprocesses has rejected (there are also logfiles in the intermediate files which will show you the same thing).

A second set of files is produced for Newbler or any other program that doesn't accept XML files. These have the bases removed. You can use them, for example to do BLASTs with or against the unassembled reads.

General tip: For BLAST you should generally make sure that there are no IUPAC codes in your data (some versions convert them to X). The script clean_iupac.pl converts a IUPAC DNA code to a ATCG code (see its perldoc).

NB: I recommend processing each technology/input dataset in a separate directory. If you have datasets from same technology but generated with different methods (e.g. vectors) or you are using GenBank derived data, then you should concatenate data from the same technology before the assembler. use the cat command

```
$ cat [fastafilename1 fastafilename2 ... fastafilenameN] > outfasta
```

For XML files, you can concatenate them with the merge_trace_xmls.pl script. As always, see its perldoc for how to use it.

Feel free to use any other subscripts in est_process for whatever you want. They all should have a perldoc describing what they do and how to use them.

Walkthrough

Let's try an example. One based on Sanger as 454 will take too long and the output is no different.

So, do you remember the Drosophila melanogaster ESTs we downloaded with srsdownload.pl ? Oh, you didn't do that... That's OK, let's do it together and see how it works, but we will download something smaller, such as *Heliconius melpomene* ESTs (a butterfly). First create a directory to download and process them. I'm assuming you are already into a 'parent working directory' where you store your work.

```
$ mkdir melpo_cdna
$ cd melpo_cdna
$ srsdownload.pl -m cDNA -s "Heliconius melpomene"
$ ls
```

It should take about 90 seconds. We will have a file 34740.cDNA.fasta contained the sequences. The first number is the NCBI taxonomy ID for our species (using IDs is safer, avoids misspellings). These should be

preprocessed (if they are in GenBank) but let's do it anyway. Because we don't know what adaptor they used (and we don't have time to check with GenBank/publications) we will use the -noadaptor option and hopefully they (actually me some years ago) got that part right.

Let's assume you don't have a config file setup yet, so everything will be in the command line. You will obviously have to correct the paths to match your system

This will remind us the options

```
$ preprocess.pl
```

```
$ time preprocess.pl -noadapt -de /db/blastdb/ecoli_pseudomonas.fsa.masked.nr -dr /db/blastdb/rDNA_nt_inv.fsa_nr -dm /db/blastdb/mito_aa_inv_80.fsa.trim -rl /db/blastdb/repeats_insecta_extra.repeats_nr95 -strain GenBank -thread 4 -deletetmp -backuptmp -archivelog -tech sanger -p Melpo_genbank 34740.cDNA.fasta
```

It will use four threads for BLAST and SSAHA (so make sure you have at least 4 CPUs available or reduce the number; you can also increase it). It will also produce a strain file with GenBank as the id. Files will have the prefix Melpo_genbank.

The time command in the beginning is completely optional. It will tell (at the end) how long it took (real time i.e. wall-clock time) and the actual CPU time (called 'user' i.e. don't include the time waiting for the hard disk to read/write). See also time -v

This will eventually (5m59sec on my computer) produce the assembler input files, a set of logfiles archived in a tarball and backup the intermediate files before deleting them. That's it!

I put the output in the test folder of est_process so you can verify against your version.

If you want to reduce the command line next time, simply edit the config file found in the same directory as the script, or produce a new one (if you want to keep track for different projects) and provide it with the -config option. The layout is a simple text file with these variables:

```
adaptor_db="/db/custom/454adaptors.fasta"
mini_adaptor_db="/db/custom/454restr_sites.fasta"
ecoli_db="/db/custom/ecoli_pseudomonas.fsa.masked.nr"
rdna_db="/db/custom/rDNA_nt_inv.fsa_nr"
mito_db="/db/custom/mito_aa_inv_80.fsa"
repeat_lib="/db/blastdb/repeats/insecta_extra.repeats_nr95"
```

TIP:

If you are using 454 data, then you must include an adaptor library. Ask your sequence center/provider. If you don't... well... you won't enjoy it the results of the assembly...

Running an Sanger and/or 454 assembly with parameterization

The most challenging process in assemblies, is knowing which parameters to use. Newbler and other black boxes, allow you to go for the default settings and get /an/ assembly but not necessarily /the/ assembly you want.

The script `parameterize_assembly.pl` helps you to do this but should not be used as a black box either. Please read the MIRA manual so you know which parameters you wish to tweak. Then the config file can be edited to allow you to produce runs with the different settings and compare them with certain statistics (please see our est2assembly paper). I do provide a config file for the brave ones amongst you.

Configuration file

The format of the config file is simple. If there is # character, then this line is ignored. then we have the basic settings which are always appended if the relevant technology is used:

```
mira_settings_basic -> Always used
mira_settings_sanger -> For sanger
mira_settings_454 -> For 454 data.
```

Then we have the parameters which you wish to test, one per line: They start with the word "parameter:" and come in pairs of key and value. The line is comma delimited (except parameter which is colon delimited)

e.g.

```
parameter:
settings, -SKIM:rt=10, settings_454, -AS:ardct=3, settings_sanger, -AS:ardct=3, assembler, MIRA, description, Repeat coverage multiplier increased to 3 repeat threshold defaults to 10
```

translates to:

- GENERAL_SETTINGS in mira will include -SKIM:rt=10
- SANGER_SETTINGS will be -AS:ardct=3
- the assembler is called MIRA
- the description in the log file is "Repeat coverage multiplier increased to 3 repeat threshold defaults to 10". The special variable \$runquality will be replaced with actual run number in the log file so just add it in.

We use reference databases to benchmark each assembly run. If you wish to use friendly names in the log output, then add lines like these:

```
database:Agambiae_ref_vbase_uniprot_100, Anopheles RefSeq VectorBase and Uniprot (nr100)
```

Reference databases

We recommend you use protein databases from the same organism you're sequencing or at least closely related (but even same phylum will work fine). For benchmarking it is not terribly important as we are interested in the relative performance rather than the absolute one. However, if your reference database is not very good (i.e. too distant) then the benchmarking will not have sufficient data to make a comparison robust.

Output

When an assembly is finished, a comma separated file can be loaded in a spreadsheet program to determine which assembly is optimal for you. Choosing an assembly is subjective (please read our paper) and depend on the aims of your project. We tend to go for those that maximise coverage of a reference proteome and maximise the number of reads even if the annotation redundancy is high. Annotation redundancy is essentially how many multiple hits your assembly has versus the reference database (totalled in both directions).

Walkthrough with *H. melpomene*

Now we can continue with our walkthrough of analysing the *H. melpomene* Sanger sequences downloaded from GenBank. The following output exists in the test folder of parameterize_assembly so you can check the output against what you will try yourself now.

make a directory (mkdir) where the assemblies will run and make a note of where your Melpo_genbank_Sanger.cleaned.fasta.x file is. For me it is at ../../.. /est_process/test/melpo_cdna/ relative to my current working directory.

```
$ time parameterize_assembly.pl -f1 ../../.. /est_process/test/melpo_cdna
/Melpo_genbank_Sanger.cleaned.fasta.x -t1 sanger -p Melpomene_assembly_genbank -accurate -config
parameterize_assembly_dbest.conf -log -newbler ../../.. /est_process/test/melpo_cdna
/Melpo_genbank_Sanger.cleaned.fasta.x.in_newbler -db /db/blastdb/silkpro\_V2\_ref\_db\_uniprot\_100
--thread 6
```

- This command will launch the parameterization of the assembly using the FASTA file we produced last time.
- It will automatically find the qual and xml files if they are simple extensions of the fasta name +.qual +.xml respectively.
- We specify that the input file f1 is from Sanger capillary technology (via t1)
- We give the project a name Melpomene_assembly_genbank and request an accurate assembly.
- We will use a default config present in the distribution that I use for dbEST (feel free to customize).
- We are asking for a log file to be kept
- Also to run newbler using the fasta file that has the masked sequence removed. Newbler will also find the quality file by itself.
- Then we are specifying which database we will use for benchmarking the different parameter (the Silkmoth proteins from RefSeq + Uniprot + SilkDb made non-redundant at 100% identity). You can include multiple

- db options, one after the other.
- Finally we are asking for 6 parallel threads (CPUs).

Tip while you wait:

You might as well go for lunch now, as this will really take a while. If you are running this from a computer connected to a server and you're afraid of losing your connection, then add screen in front (if it is installed on your server). This will put the command on a separate process in the background. You can later fetch it via screen -r

```
$ screen time parameterize_assembly.pl -f1 ../../../../est_process/test/melpo_cdna
/Melpo_genbank_Sanger.cleaned.fasta.x -t1 sanger -p Melpomene_assembly_genbank -accurate -log
-config parameterize_assembly_dbest.conf -newbler ../../../../est_process/test/melpo_cdna
/Melpo_genbank_Sanger.cleaned.fasta.x.in_newbler -db /db/blastdb/silkpro V2 ref db uniprot 100
>Ctl A then Ctl D
```

The last key combination will detach the screen which carries on the background. Later, to connect (if it is still running):

```
$ screen -r
```

If it has finished then the screen closes automatically and you cannot see any output (including errors). This can be ok if you don't expect any. If you might then use screen first to create the 'pseudo-terminal' and then launch the command;

```
$ screen
$ time parameterize_assembly.pl -f1 ../../../../est_process/test/melpo_cdna
/Melpo_genbank_Sanger.cleaned.fasta.x -t1 sanger -p Melpomene_assembly_genbank -accurate -log
-config parameterize_assembly_dbest.conf -newbler ../../../../est_process/test/melpo_cdna
/Melpo_genbank_Sanger.cleaned.fasta.x.in_newbler -db /db/blastdb/silkpro V2 ref db uniprot 100
> Ctl-A then Ctl-D
```

You can use screen -r to go back.

Next time you use this program, you might want to include a 454 dataset. simply give -t1 as 454, or if you use both sanger and 454 then keep -t1 sanger and add -t2 454 -f2 <your 454 filename from preprocess>

Continuing with our walkthrough:

Once the assemblies have all finished, we can look at the output:

```
$ less Melpomene_assembly_genbank.accurate.blast_analysis.csv
```

or you can use a spreadsheet program.

A summary of Melpomene_assembly_genbank.accurate.blast_analysis.csv is available as Melpomene_assembly_genbank.accurate.blast_analysis.total.csv. It has only the lines with Total and is applicable if you used more than one reference proteome.

How you choose the assembly is subjective (see above) but I think

Melpomene_assembly_genbank.accurate.2 is pretty good (if you use the same version of MIRA you should get the same result). You can also see that Newbler does not perform very well (hopefully someone in 454 Life Sciences reads this!).

I personally either delete all the runs I do not want

```
$ rm -rf <directory>
```

careful this is a very powerful command : it does not ask for confirmations and it deletes directory structures: in Linux if you delete smth it's gone! (but see *libtrash*).

Did I mention you should be careful with `rm -rf` ?

the other option I use when I want to keep the runs (just in case) and there is no problem with disk space is to make a soft-link, i.e. a shortcut e.g.

```
$ ln -s Melpomene_assembly_genbank.accurate.2 chosen
or if you want the result directory:
$ ln -s Melpomene_assembly_genbank.accurate.2/Melpomene_assembly_genbank.accurate.2_d_results/
chosen
```

then you know that chosen is your assembly of choice and you don't have to remember which one it was.

Tip:

An assembly is not a clustering process. Please make sure you understand this principle while working with EST projects.

Assembly annotation

BLAST annotation

Once an assembly is chosen, it should be annotated so you data mine it. First you have decide what kind of databases you wish to annotate with. Then make the relevant BLAST files.

Also you might want to consider to add Enzyme Classification, Gene Ontology and KEGG pathway terms using a similarity to an annotated protein. In order to do this, you will need to predict a protein for each contig.

Walkthrough for BLAST

Let's continue with annotating our Melpomene data. I put the examples in the analyze/test directory.

You have two options: running `blastall` directly (make sure it is installed) or using the `analyse_assembly.pl` script. The latter option does some extra work

(and takes extra time) as it calculates the b.p. coverage and other stats (same method as in the parameterization) and has the option to extract the parts of the query which have a hit (the `-extract` option) in case you want to look at it.

Let's produce an analysis using the known proteins (uniref100 from UniProt):

let's make a local link to the input file which will also be renamed to smth more friendly (to make future commands shorter). (in your system the input file might have a different path!)

```
$ ln -s ../../../../parameterize_assembly/test/melpo_cdna_assembly/chosen
/Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta Melpo_assembly.fsa
$ analyse_assembly.pl -i Melpo_assembly.fsa -db "/db/blastdb/uniref100.fasta" -t 10 -ce 1e-5 -cs
80 -extract
```

Note that you have to give your own `-db` argument for uniref100 (also the quotes are not necessary). Also make sure you have enough CPUs for the `-t` argument (threads). BLASTx versus Uniref100 will take a (very) long time so feel free to use a smaller database if you prefer (uniref90, uniref50 or even the reference proteome BLASTx you have already completed during the assembly). My computer took 207m36.065s for the above command.

The `-cs` and `-ce` arguments give cutoffs for score and e-value (which do not correlate with each other). It is quite stringent but a bit-score lower than 80 will produce false alignments. Lower it if wish. The BLAST report itself is actually run with lower cut-offs (1e-1) The downside of this is very large BLAST reports (0.6 Gb for the above one) but it does allow investigation of lower cut-offs. The HASH used for the analysis is read with the specified options above. This means you can use *analyze_blast.pl* change the cut-offs without having to re-run the BLAST (or re-write the hash; use the `-uh` option).

e.g.

```
$ analyze_blast.pl -cs 60 -ce 1e-1 -uh Melpo_assembly.fsa.x.uniref100.fasta.refblast
```

The `.x` in `Melpo_assembly.fsa.x.uniref100.fasta.refblast` appears because the input file converted all IUPAC codes to a valid nucleotide A,T,C,G as BLAST is notorious for not liking them (I believe maybe fixed in a new version, but better to be sure). You can switch off this behaviour with the `-noclean` argument of *analyse_assembly.pl*

Also note that the analysis of the BLAST is not necessary for the protein annotation below. It does provide important information about your assembly though.

NB. *analyse_assembly.pl* compresses the BLAST report after it is finished with it (to save space). So if you need, please uncompress it with *bunzip2* again.

```
$ bunzip2 Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta.x.uniref100.fasta.refblast.bz2
or
$ bunzip2 -k
Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta.x.uniref100.fasta.refblast.bz2
```

to keep both the BZ2 and BLAST files.

Let's also create BLASTs against a mitochondrial and rDNA database (one for Insects is included in distribution). No need to analyse it so just use the normal blastall executable.

```
$ blastall -i Melpo_assembly.fsa -d "/db/blastdb/mito_aa_inv_80.fsa.trim" -Q 5 -o mito.bls -p
blastx -a 5
$ blastall -i Melpo_assembly.fsa -d "/db/blastdb/rDNA_nt_inv.fsa_nr" -o rna.bls -p blastn -a 5
```

The warning about the `-threads` option applies here too for the `-a` option.

Protein prediction

Please do note that not all contigs will be from coding regions (an assumption frequently made and an honest mistake) so forcing a protein prediction may result in an utter unrealistic one. Nonetheless, a protein prediction for the whole assembly is recommended and we use a program called prot4EST. A version I have made a few tweaks on is included in this distribution.

The program (you should also read its paper...) relies on two main approaches that work and two that generally don't : similarity to known protein, ESTScan, Decoder, longest ORF. A contig is first tried with the first two and if these fail it tries with the other two. ESTScan requires a organism-wide codon-usage table (the concept of which is arguable to be realistic as loci can utilise different codon usage but let's ignore this now). How we produce it, we will talk about once we cover the similarity step.

We suggest you run a BLASTx versus Uniref100 from Uniprot. The full blast report must be provided in the configuration file of prot4EST (see the prot4est directory). Using Uniref100 will make it more likely you pick a similar hit than using a reference proteome (as you did before) but it will be much much slower (because parsing such reports and tiling the high scoring pairs (HSPs) in a hit is rather slow). Any proteins predicted with this method are almost certainly real. Prot4EST uses an internal cut-off score so feel free to use whatever cut-off you think is real. Once you have this BLAST report you can use it to build a codon and HMM model for ESTScan.

- The easy way to build one is if you had a lot CDS data in GenBank for your species, but that is probably unlikely if you are trying to predict proteins from ESTs. If you have used the `srsdownload.pl` option, then you probably have already included those CDS in the assembly.
- There is the 'hard' way which is also more reliable but also made easy due to a script I provide, it pretty much automates the whole procedure.
 - We will build a set of reliable proteins using the BLASTx report from above; even if they are partial they will be enough to build a model of codon usage and a HMM for ESTScan.
 - The HMM model is build using a reference proteome which is then backtranslated using the codon usage table created.
 - So in effect, we create fake CDS regions based on known proteins using your species' codon usage model (as determine through your

assembly).

See the options:

```
$ ic_build_codons_model.pl
```

There will be two files of importance: the .cod is the codon usage table and .smat is the HMM model for ESTScan. Edit the prot4EST file to include these two.

To speed things up, you also need to do a BLASTn versus rDNA genes and a BLASTx versus mitochondrial genes. Be careful; when you do the BLASTx versus mitochondrial genes, you should make sure you provide the correct codon table number using the -Q option of blastall (e.g. -Q 5 for insects). Name these output files rna.bls and mito.bls respectively and let them reside on the working directory where you will run prot4EST from (and where the config file resides). See the walkthrough about BLAST above.

Walkthrough for protein prediction

So let's predict proteins for our Melpomene data: I will use the prot4EST/test/melpo_p4e directory to store my results so you can compare.

Create symlinks (paths may differ; don't forget the dot as target at the end):

```
$ ln -s ../../analyze/test/melpo_annotation/rna.bls ../../analyze/test/melpo_annotation/
mito.bls ../../analyze/test/melpo_annotation/Melpo_assembly.fsa.x.uniref100.fasta.refblast
../../analyze/test/melpo_annotation/Melpo_assembly.fsa.x <path to
assembly>/Melpo_assembly.fsa.qual .
```

Make a copy the empty p4e.config file

```
$ cp ../../p4e.config .
```

Make a dir for our codon modelling and build it

```
$ mkdir codons
$ cd codons
$ ic_build_codons_model.pl -b Melpo_assembly.fsa.x.uniref100.fasta.refblast -i Melpo_assembly.fsa
-a "/db/blastdb/silkworm/silkpro_ref_uniprot.clean_nr90" -s "Heliconius melpomene"
```

Finally you need to create a special directory proteins.x with FASTA sequence and quality where the files are split (using splitfasta.pl) for each contig (from the assembly). Please use the unpadded files and make sure the sequence has no IUPAC codes (prot4EST will break):

using the linked files from above:

```
$ splitfasta.pl -i Melpo_assembly.fsa -suffix seq -dir proteins.x
$ splitfasta.pl -i Melpo_assembly.fsa.qual -suffix qlt -dir proteins.x
```

Once you have all the files ready, edit p4e.config and point to the correct files. Then run it with this command (from the directory containing the p4e.config).


```
$ prot4EST_2.3-AP.pl 1
```

The 1 option is to make the process automatic. You might want to use screen again, as this will take also a long time (due to parsing the Uniref100 BLAST file using BioPerl). It took 30' on my computer.

Proper naming

This step can be run at any time until now, but it should be run before the use of annot8r next.

As we are going to database our data, it is important to provide a standard approach to naming them. The naming scheme recommended (and a script provided for) is this:

Generally it looks like this

<a unique database ID of your choice><species ID><Assembly identifier><data type><serial number>

Specifically, we are going to use:

- For an **assembly contig**: <two letters for db id><NCBI Taxid number><assembly run where Aa is 1 Ab 2 etc><E for expressed><con for contig><serial number>
- For a **peptide/ORF**: <two letters for db id><NCBI Taxid number><assembly run where Aa is 1 Ab 2 etc><A for automated prediction><pep for peptide or orf for open reading frame><serial number>
- For a **marker**: <two letters for db id><NCBI Taxid number><assembly run where Aa is 1 Ab 2 etc><M for marker><snp for SNP, ssr for SSR etc><serial number>

The DB id should be the same in your Lab. We use IC (for InsectaCentral). The assembly run is meant to be used when new assemblies are created (by adding new data or updating something) not by the assembly parameter set chosen.

This naming scheme ensures that each object is completely unique and can have exist even after an assembly is retired. If needed, link tables can provide such historical links.

Please also note, that the protein/orf serial id follows each other but not the contig id. This enables us to store multiple protein predictions for the same contig. The prot_main.psql.named link table in the prot4EST output directory provides the links between them (and is used by subsequent steps such as SNP alignment).

So let's name our assembly file (in CAF format), the FASTA files (the unpadded sequence/quality files and the padded quality for the SNP analysis later), the BLAST reports and the protein results. All except the later involve a simple

pattern search.

Renaming the prot4EST includes a special feature not available in prot4EST 2 but will become available in prot4EST 3 (I don't know the release date, so please don't ask me...). As you probably know (because you read the prot4EST paper), it works by predicting and correcting the protein translation (due to frameshifts). As a result the open reading frame is not simply a sublocation within the contig. For that reason `ic_create_naming.pl` will try to find the correct sequence of the ORF by trying to align the protein with the contig using `fasty34` (by Pearson) and if that fails by using `backtransambig` (from EMBOSS) and correcting the ambiguous nucleotides where possible using the contig. This can take quite a long time for 454 assemblies and I know it is not the best implementation but as the next version of prot4EST is due to implement this, I saw no reason taking it apart and doing it myself.

We will use DM for database name (for DeMo). It will be the first assembly we named and made public, so we don't have to set `-assembly 1` (it's the default). If we know the NCBI taxid we can give it in the `-species` option directly, instead of writing "Heliconius melpomene".

The output files can be specified, the default is the original file `+.named`.

Go to the directory where the assembly is and rename the Assembly

```
$ ic_create_naming.pl -t caf -d DM -s "Heliconius melpomene"
```

```
Melpomene_assembly_genbank.accurate.2_out.caf
```

Rename the FASTAs

```
$ ic_create_naming.pl -t fasta -d DM -s "Heliconius melpomene"
```

```
Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta
```

```
Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta.x
```

```
Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta.qual
```

```
Melpomene_assembly_genbank.accurate.2_out.padded.fasta.qual
```

Go where the BLASTs are. Rename the BLASTs.

```
$ ic_create_naming.pl -t blast -d DM -s "Heliconius melpomene"
```

```
Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta.x.uniref100.fasta.refblast
```

Go to where you run prot4EST from. Rename the prot4EST output. The `-type` is `p4e` and the input will be the output directory where the files have been produced (on my computer this took about one minute)

```
$ ic_create_naming.pl -t p4e -d DM -s "Heliconius melpomene" output/
```

Again you will see a fasta file with an X: `translations_xtn.fsa.X`, `translations_xtn.fsa.X.named`. Prot4EST sometimes produces ambiguity codes (e.g. B) which are not recognised by many software. These amino acids have been converted to X (i.e. unknown).

you can rename your files to match the header:

```
$ head translations_xtn.fsa.X.named
```

```
$ mv translations_xtn.fsa.X.named DM34740AaApep.fsa
```

Annotate with EC/GO/KEGG and walkthrough for annot8r

Now we can use the proteins predicted by p4e to BLAST versus known proteins which have an EC, GO or KEGG term. The relevant postgres databases are included in the annot8r directory, along with a hacked copy of annot8r (again, you should read their paper and use their software to understand what it does).

If you want to create new Uniprot FASTA files or new psql databases, then you need to use the original annot8r software. I provide, only for convinience, the psql and BLAST databases that were current on 01Jun2009.

Go to the annot8r directory. Extract the psql files

```
$ bunzip2 *psql.bz2
```

Assuming your postgres (psql) installation is working correctly, if you have more than one postgres cluster (unlikely if you are a novice user)

```
$ pg_lsclusters
```

And make note of the port number you wish to use. You will have to provide it as -dbport for annot8r or -p for psql. Likewise for host (with -dbhost or -h respectively.)

Load your psqls

```
$ psql < a8r_ecbase.psql
```

```
$ psql < a8r_gobase.psql
```

```
$ psql < a8r_keggbase.psql
```

And three new databases, a8r_ecbase, a8r_gobase and a8r_keggbase will appear in your psql cluster.

Overview of how to use it is, as always, available with perldoc:

```
$ perldoc annot8rAP-gff.pl
```

Essentially, you are taking a protein FASTA file, BLASTing it against known proteins with KEGG, EC, GO annotation and based on a particular cut-off you can assign KEGG, EC or GO terms using the IEA (Inferred Electronic Annotation) code.

Annot8rAP-gff.pl needs as input BLAST files. Produce them with the method of your choice using each of the databases (which you will need to uncompress first)

In the annot8r directory:

```
$ tar -xjf uniprot_GO.fsa.tar.bz2
```

```
$ tar -xjf uniprot_KEGG.fsa.tar.bz2
```

```
$ tar -xjf uniprot_EC.fsa.tar.bz2
```

BLASTP them with your protein translations,

e.g.

```
$ blastall -i DM34740AaApep.fsa -o DM34740AaApep_vs_EC.blastp -e 1e-5 -d "/db/blastdb/uniprot_EC.fsa" -a 2 -p blastp -b 30 -v 30 2>errors
```

```
$ blastall -i DM34740AaApep.fsa -o DM34740AaApep_vs_KEGG.blastp -e 1e-5 -d "/db/blastdb/uniprot_KEGG.fsa" -a 2 -p blastp -b 30 -v 30 2>errors
```

```
$ blastall -i DM34740AaApep.fsa -o DM34740AaApep_vs_GO.blastp -e 1e-5 -d "/db/blastdb/uniprot_GO.fsa" -a 2 -p blastp -b 30 -v 30 2>errors
```

The `2>errors` will redirect any errors to the errors file. Feel free to check it between BLASTs. Make sure the `-a` is optimized for your system and the `-d` argument is correct. Expect the BLAST vs GO to take quite some time (hours). We will also estimate some parameter/statistics for each protein with the `annot8r_physprop.pl` script

Then simply

```
$ annot8rAP-gff.pl DM34740AaApep_vs_EC.blastp DM34740AaApep_vs_KEGG.blastp
DM34740AaApep_vs_GO.blastp
$ annot8r_physprop.pl DM34740AaApep.fsa
```

Annot8r will ask the postgres databases you made in the last step in order to decode the hits and then produce an annotated file ready to be converted to GFF (see next chapter).

Annotate with InterProScan

Please see the LSF section and the InterProScan manual as how to use someone else's program is not the aim of this manual. InterProScan can be tricky to set up and takes an awful long time to run (up to 10 min per protein) so please use only if you are an advanced user. We highly advise the use of PC-Farm but I must caution you: please work with your system administrator as `iprscan` can IO starve the servers (and therefore crash them).

Once you produced an `iprscan` result in raw format (it must be in raw) then you can give it as argument to the GFF creation program later on (along with the other `annot8r` output files).

Annotate with SNPs

Single Nucleotide Polymorphisms are very important for a variety of reasons (which I will not delve into). One way a SNP can shown is as a polymorphic base in the consensus (using the IUPAC nucleic acid codes). The assembler MIRA is producing such SNP output and converting it to 'Tags' which the `caf2gff` script later on will allow you to extract this information. For many applications, however, a more stringent search may be required, and therefore I provide a method to detect SNPs de-novo from the assembly (I call them high-quality predicted SNP): the polymorphic nucleotide has to exist in at least two or three reads and have an certain number of invariable positions up- and down-stream of the SNP (called padding). The script that automates the process is called `ic_create_snps.pl`.

Walkthrough for SNPs

Go to the directory where the assembly resides, (especially the CAF file) and use the following command.

NB: Due to the programs used it will take quite a long time, especially with 454 assemblies. There is a chance that I will rewrite the script to not make use of SEAN and be a bit faster, but at the moment this is not a priority.
NB2: You will probably wish to use the .named files in order to basecall the SNPs, but you don't have to. It can be done in the GFF stage

```
$ cd parameterize_assembly/test/melpo_cdna_assembly/chosen
$ ic_create_snps.pl -caf Melpomene_assembly_genbank.accurate.2_out.caf.named -cov 2 -padd 20
```

This will identify SNPs if the minor allele occurs at least two times and there are 20 bp invariable up and downstream. The script will use `convert_project` from MIRA to create a FASTA file of all the ESTs that exist in the assembly (you can also provide the original EST FASTA file if you don't want to wait, the script looks for the `<caf filename> + the .fasta suffix`), and then it creates an ACE file (to use with SEAN). It splits the ACE file to one ace per contig and runs SEAN on each one.

A simple text file called "snps" containing all the SNPs for each contig will be printed out but this will not be very informative due to the method SEAN uses to parse the files. You will need to make the GFF file, which brings us to the next section.

Creating GFF files

In order to distribute or use the information you just created in a more standard format we will make use of the GFF file format (read <http://gmod.org/wiki/GFF>). `est2assembly` contains a number of tools to create valid GFFs for Chado or Bio::DB::SeqFeature format from all the data types you have generated so far.

So let's look at the `gffcreation` folder and our Melpomene test

go to `gffcreation/test/melpo_gffs` and we will create a GFF for each file. By not providing the `-o` (output) option, all GFFs will be created in the current dir and will be named as: `<input file> + .gff` (and also `+.biofeature` for BioFeature compatible files).

Creating a GFF of the **assembly** is straightforward and we support both contig-read and read-contig formats:

```
$ ic_caf2gff.pl -e -org "H.melpomene" -biofeature "../../parameterize_assembly
/test/melpo_cdna_assembly/chosen/Melpomene_assembly_genbank.accurate.2_out.caf.named"
```

Likewise, the GFF for **BLAST** is straightforward (but for Uniref100 in 454 assemblies, it will take a long time - due to opting for the robust BioPerl processing method...).

```
$ ic_blast2gff.pl -cs 80 -o "H.melpomene" -biofeature -l 10 "../../analyze
/test/melpo_annotation/Melpo_assembly.fsa.x.uniref100.fasta.refblast"
```

The **Protein Prediction** GFF is straightforward to make. You need the output

of prot4EST and the ic_create_naming.pl.

```
$ ic_p4e2gff.pl -main "../../../prot4EST/test/melpo_p4e/output/prot_main.psql.named" -blast
"../../../prot4EST/test/melpo_p4e/output/prot_hsps.psql.named" -pt "../../../prot4EST
/test/melpo_p4e/output/DM34740AaApep.fsa" -nt "../../../prot4EST/test/melpo_p4e/output
/orf.all.fsa" -e -o "H.melpomene" -biofeature
```

For the SEAN output, we need to give the directory where the splitted ACE files reside and the file containing the quality values (which must be padded, i.e. align to the assembly). In order to identify if a **SNP** is coding and synonymous/non_synonymous, we also need some of the protein prediction files.

```
$ ic_sean_snp2gff.pl -dir "../../../parameterize_assembly/test/melpo_cdna_assembly/chosen
/Melpomene_assembly_genbank.accurate.2_out.caf.named.ace_dir" -org H.melpomene -qual "../../../parameterize_assembly/test/melpo_cdna_assembly/chosen
/Melpomene_assembly_genbank.accurate.2_out.padded.fasta.qual.named" -orf ../../../prot4EST
/test/melpo_p4e/output/orf.all.fsa -link ../../../prot4EST/test/melpo_p4e/output
/prot_main.psql.named -pad 20 -table inv
```

For **annot8r** and **iprscan**, you will need to provide the physical properties outfile (.physprop) as a link and then you may specify one or more GO, EC, KEGG, iprscan outfiles. There is an extra experimental option for those expert users using chado: you can make use of the ontologies by specifying the -dsn option to a chado connection file (see perldoc). Otherwise, if using SeqFeature, specify -nodb. You can also specify the cut-offs (score and eval) to be used. If you wish to use different one for KEGG, EC, GO etc then you should run them separately.

```
$ ic_annot8r2gff.pl -nodb -cs 80 -ce 1e-10 -phys ../../../annot8r/test/melpo_annot8r
/DM34740AaApep.phys ../../../annot8r/test/melpo_annot8r/DM34740AaApep.fsa_1.in.iprscan ../../../
/annot8r/test/melpo_annot8r/DM34740AaApep_vs_EC.blastp.annot.EC ../../../annot8r
/test/melpo_annot8r/DM34740AaApep_vs_GO.blastp.annot.GO ../../../annot8r/test/melpo_annot8r
/DM34740AaApep_vs_KEGG.blastp.annot.KEGG
```

General tips

Please note

- 1) Make sure you use the .named files! Without having standard ID, we cannot create GFFs.
- 2) for p4e you will notice we have many options. The -main file links the protein objects with the contigs. The -blast is not a BLAST report but the results of the BLAST parsing from prot4EST.
- 3) The -e option allows you to embed a copy of the FASTA sequences at the end of the GFF
- 4) Parsing uniref100 is quite slow due to all the HSP objects that BioPerl has to parse...

Populating a SeqFeature database for GBrowse

This step assumes you know enough about databases in order to use them confidently, so please note this is for advanced users.

The current version of est2assembly is aimed for users who wish to be able to analyze and display their transcriptomic data. It allows integration of GBrowse (and other applications) with Chado but we do not recommend it as it is very slow. In addition, the current version does not leverage the full data warehousing capabilities of Chado. Because we are building such a warehouse in our lab, do expect the next version to have this capability but please do note that chado has a steep learning curve and we don't recommend anyone but those versed in Bioinformatics to utilize it (e.g. if you don't know what a DAG is, you will have difficulties to populate chado).

The database schema of Bio::DB::SeqFeature is much easier to understand, use and much, much faster. It does not offer (or is meant to) the data warehousing capabilities of Chado but it is sufficient for the purpose of viewing data.

As mentioned previously, there are two types of GFFs, those with the biofeature tag in the filename and those without. The latter can be uploaded in chado with the following command

```
$ gmod_bulk_load_gff3.pl --noexon --dbname $DB_CHADO_NAME -dbhost  
$CHADO_DB_HOST -dbport $CHADO_DB_PORT --recreate_cache --analysis  
--organism "name of organism from organism table" -g gfffile
```

for SeqFeature things are easier using a script from the -dev branch of BioPerl:

See the help

```
$ bp_seqfeature_load.pl -h
```

or

```
$ perldoc bp_seqfeature_load.pl
```

I recommend using it with a postgres database (as mysql does not have the data integrity abilities of postgres). The command line can be quite tricky and it is tedious to load many GFFs. So I provide the script ic_loaddata.pl to help you.

```
$ perldoc ic_loaddata.pl
```

You can provide as many CAF- (assemblies), p4e- or BLAST-derived GFF files you want. For each CAF there must be a p4e file. The script will (re)create a SeqFeature database, making a backup if necessary. It will use your /tmp as a

temp directory to load the GFFs more quickly.

GBrowse integration:

In the gbrowse folder of this distribution there are some .inc files which you should customize for your system. If you copy them to your GBrowse 1.69 configuration folder (e.g. "/etc/apache/gbrowse.conf/") and pass the name of the directory to ic_loaddata.pl with the -config option, then you will create GBrowse 1.69 configuration files for each set of CAF/p4E file set. For each set you will have a _caf, a _orf and _prot .conf file, one for each page of GBrowse that shows the assembly, the ORF object and the protein object respectively.

So the command for our melpomene data (in one simple line) is:

From the gffcreation/test/melpo_gffs directory:

```
$ ic_loaddata.pl -caf Melpomene_assembly_genbank.accurate.2_out.caf.named.biofeature.gff -p
DM34740AaApep.fsa.biofeature -b
Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta.x.uniref100.fasta.refblast.named.gff.biofeatu
DM34740AaApep.phys.gff DM34740AaApep_vs_EC.blastp.annot.EC.gff
DM34740AaApep_vs_G0.blastp.annot.G0.gff DM34740AaApep_vs_KEGG.blastp.annot.KEGG.gff
DM34740AaMsnp.gff DM34740AaApep.fsa_1.in.iprscan.gff -prefix dm_ -create -a "/etc/apache2
/gbrowse.conf/"
```

NB:

- You can use more than one -c -p -b -s -t option or you can specify many files one after the other in one -b etc option. This applies to any option which has the symbol {,} in the PerlDoc or usage instructions.
- If multiple assemblies are specified, however, you have to follow the same order in each of the -c -p -s.
- The -t option is optional, as it will be activated if you decide to load a p4e and/or caf file. You can use it to recreate the config files without loading new data.
- The -b option can handle any annotation: BLAST, SNPs, annot8r etc.
- the -a value might be different in your system.

So now if you use

```
$ psql -l
```

you should be able to see your database as dm_34740. I included a backup of this file using pg_dump :

```
$ pg_dump dm_34740 -Cx0f dm_34740.psql
```

where -x and -O are used to remove personal info relating to my system, -f specifies the file and -C is used to include the creation commands in the .psql file so that you can restore the backup like this:

```
$ psql -f dm_34740.psql
```

That's it! Now you have a database and the GBrowse config files:


```
$ ls -trl "/etc/apache2/gbrowse.conf/"

-rw-r--r-- 1 alexie alexie 35979 2009-06-27 18:25 34740_prot.conf.bak
-rw-r--r-- 1 alexie alexie 35979 2009-06-27 18:25 34740_prot.conf
-rw-r--r-- 1 alexie alexie 4936 2009-06-27 18:25 34740_orf.conf
-rw-r--r-- 1 alexie alexie 28675 2009-06-27 18:25 34740_caf.conf.bak
-rw-r--r-- 1 alexie alexie 28675 2009-06-27 18:25 34740_caf.conf
```

Where .bak will exist if you run it more the once and it holds your backup. Each of these .conf files is scanned by GBrowse when it starts so it should be available under your Gbrowse installation as /34740_caf or /34740_prot or /34740_orf for the assembly, protein and ORF pages respectively.

The assembly _caf page links to the ORF _orf page which links to the protein _prot page. My demonstration is available here:

http://rfc.ex.ac.uk/cgi-bin/gbrowse/34740_caf/

When you click on it, you will get an empty GBrowse because no contig has been specified. But look at the tracks and compare with the prot and orf pages:

http://rfc.ex.ac.uk/cgi-bin/gbrowse/34740_prot/

http://rfc.ex.ac.uk/cgi-bin/gbrowse/34740_orf/

This manual is not a tutorial for GBrowse, but very briefly:

What we need to do is specify a contig (e.g. DM34740AaEcon1), protein (e.g. DM34740AaApep515) or ORF (e.g. DM34740AaAorf515) to get a view. This can be given in the URL or in the search box ("Search Landmark or Region:")

http://rfc.ex.ac.uk/cgi-bin/gbrowse/34740_caf/?name=DM34740AaEcon1

http://rfc.ex.ac.uk/cgi-bin/gbrowse/34740_prot/?name=DM34740AaApep515

http://rfc.ex.ac.uk/cgi-bin/gbrowse/34740_orf/?name=DM34740AaAorf515

Let's look more closely how you interrogate the dataset using GBrowse in the next section.

Interrogating the dataset and understanding the structure

For this tutorial, you can use my demonstration website or the one you built for yourself using the test Melpomene data. Let's read also the GFF files (using your favourite editor/viewer) so that we understand how things are linked. First, the _caf assembly.

```
$ less Melpomene_assembly_genbank.accurate.2_out.caf.named.biofeature.gff
```

If you don't know what the GFF file format is, see here:

<http://song.sourceforge.net/gff3.shtml> (please note, the Sanger website is very much out of date).

Each line represents a feature which is anchored on the object specified in the left hand side column (the "reference"). Each feature has an ID, which needs to be unique in the whole database (i.e. across all .gff files loaded). This ID can be used in the Search box of GBrowse:

Try giving **EE743470** to the assembly page search box. It will automatically find the contig that contains this EST ID and you will get the EST shown and highlighted.

Now let's look at some annotation: Activate the **Known Proteins** track under **Annotation**. You will get the protein match that was included in your Uniref100 BLAST. Let's look at the GFF file:

```
$ less
```

```
Melpomene_assembly_genbank.accurate.2_out.unpadded.fasta.x.uniref100.fasta.refblast.named.gff.biofeat
```

You will see the same hits in the file where the left hand side column (the reference) is the contig ID DM34740AaEcon2. Each line can have in the third column match_part or protein_match. The latter is the ueber-entity which groups the former, i.e. the HSP and HIT of the BLAST respectively. Let's pick one that has more than one HSP and let's search using the information contained in the annotation field (the last column on the right called 'attributes'). E.g. let's look at this line:

```
DM34740AaEcon3 BLASTX_uniref100 protein_match 268 417 4e-05 + .
ID=DM34740AaEcon3:uniref100:Hit1;Alias=Similar to
UniRef100_UPI0000DA399F;Dbxref=uniref100:UniRef100_UPI0000DA399F;
Name=DM34740AaEcon3:uniref100:Hit1;Note=PREDICTED: hypothetical protein n%3D1 Tax%3DRattus
norvegicus RepID%3DDUPI0000DA399F;algorithm=BLASTX;fraction_of_conserved_positions=0.0467;
fraction_of_identical_positions=0.0327;hit_rank=2;length=643;logical_length=227;
organism=H.melpomene;score=4e-05;top bit_score=52.0;top raw_score=123;total significance=4e-05
```

which has many match_parts (HSPs) following it. Look at the Note field and note the funny % characters: These are 'escape values' for URLs. They are not human-friendly but they are important for the safety of your server and will be translated in your browser. Now look at the Alias tag. It contains a protein ID from UniProt (**UniRef100_UPI0000DA399F**). The same ID is included in the Note field, with the UniRef100 ID stripped out (**3DDUPI0000DA399F**). Copy paste **3DDUPI0000DA399F** to your assembly search box and click search. You will get a list of all contigs that have this annotation. You can do this, with any part of the Note field, such as **Rattus norvegicus**. Now click to one of the search results and you will go to the contig view containing this hit.

Move your mouse over the hit ("hover") and you will get a description in a little balloon. Click and you will go to a detailed description which has a stable URL you can email to people:

http://rfc.ex.ac.uk/cgi-bin/gbrowse_details/34740_caf?name=DM34740AaEcon1%3Auniref100%3AHit1;class=Sequence;ref=DM34740AaEcon1;start=280;end=429;feature_id=7546

(Note the funny % characters, they are for your safety too). On this page you will get some links. They will point to the Uniref database for this protein.

http://www.uniprot.org/uniref/?query=UniRef100_UPI0000DA399F&sort=score

Cool huh? I thought so too... (Thank you and well done Lincoln and Scott!).

So let's look at some Markers now. The only marker module included in est2assembly is SNPs (maybe others such as SSRs in another release). Here is the GFF we produced for it:

```
$ less DM34740AaMsnp.gff
```

You will note contig 107 (DM34740AaEcon107) has a lot of SNPs. So let's look at it...

http://rfc.ex.ac.uk/cgi-bin/gbrowse/34740_caf/?name=DM34740AaEcon107
and activate the **SNP track** under **Predictions**.

You will get Pie Charts of the various alleles and if you hover your mouse you will see some more information. If you click you can see the details.

If you remember, we give a unique ID to each object, so our SNPs will also be unique and we can use it to search. So let's take **DM34740AaMsnp2** search for it. You will see that it is also aligning with the ORF so it must be coding. One way to access the ORF page is by clicking on the predicted protein shown in the assembly view.

NB: The SNP module is experimental, the main reason being that I find prot4EST (a.k.a. p4e) to be a great program but limited and buggy. I've been promised an excellent version 3 which will solve many problems so please be patient (as am I... but there is always the possibility I find time to reinvent the protein prediction wheel).

One nice thing about the way we set up our naming scheme and database is that we can have many protein (and thus ORF) predictions for one contig. You will notice that the name of the ORF, DM34740AaAorf337 is not similar to DM34740AaEcon107. The mapping was done during ic_create_naming.pl creating the link file prot_main.psql.named. This info is now stored in the GFF:

```
$ less DM34740AaApep.fsa.biofeature
```

You will notice that the protein predictions come in groups of four:

```
DM34740AaEcon59 placeholder      ORF      88      504      .      +      0
ID=DM34740AaAorf439_0;Dbxref=InsectaCentral:DM34740AaAorf439;Name=DM34740AaAorf439_0;conf_end=459;
conf_start=88;end_frame=1;ext_end=504;gene_location=nuclear;method=similarity;
organism=H.melpomene;start_frame=1
DM34740AaAorf439      prot4EST      ORF      1      417      .      +      0
ID=DM34740AaAorf439;Dbxref=InsectaCentral:DM34740AaAorf439;Name=DM34740AaAorf439;Note=Open
Reading Frame of protein prediction.;conf_end=459;conf_start=88;end_frame=1;ext_end=504;
gene_location=nuclear;method=similarity;organism=H.melpomene;start_frame=1
```

```

DM34740AaApep439      prot4EST      polypeptide      1      139      .      +      0
ID=DM34740AaApep439;Dbxref=InsectaCentral:DM34740AaApep439;Derives_from=DM34740AaEcon59;
Name=DM34740AaApep439;gene_location=nuclear;method=similarity;organism=H.melpomene
DM34740AaAorf439      placeholder      polypeptide      1      417      .      +      0
ID=DM34740AaApep439_0;Dbxref=InsectaCentral_DM34740AaApep439;Name=DM34740AaApep439_0;Note=HSP
similarity to uniref100 was 4e-50;gene_location=nuclear;method=similarity;organism=H.melpomene

```

The first feature is putting an ORF in respect to the assembly page (Econ is reference), the second feature initiates an ORF page (reference column is equal to the ID), the third does likewise for the protein object and finally the fourth links the protein object with the ORF object. We don't have to initiate the Econ (contig) object because this has been done in the .caf GFF file (take a look!).

So Econ59 has Aorf439 and therefore also Apep439 (Aorf and Apep serials also follow each other). If we wanted to add another protein prediction, it can be very easily done by finding out what is the next free unique ID for Aorf/Apep and giving it to the `ic_create_naming.pl` script when we process the p4e data. That way if for example we use different HMM for ESTScan (tier II of p4e) or database for similarity searches (tier I of p4e) then we can have both predictions sets appearing on the Gbrowse interface.

So let's look at a protein. Click on the protein track on the ORF page or just go to the protein URL:

http://rfc.ex.ac.uk/cgi-bin/gbrowse/34740_prot/?name=DM34740AaApep439

You will see that there are quite a few annotations here which do not appear in the contig page. This is because KEGG, GO, EC and InterProScan is run using proteins, not EST contigs. The main reason is that 6 frame translations of EST contigs (i.e. BLASTX) are notorious for being a) just terrible due to frameshifts decreasing the scores b) computationally 6 times more expensive than a BLASTP (which is not a big problem if you are interrogating 1000 contigs and have spare machines doing nothing but a huge problem when you're annotating hundreds of thousands or your computational resources are limited).

As usual, hovering your mouse and clicking gives more details for each annotation. These annotations are derived from the `annot8r` GFF so let's look at one of them (e.g. the GO).

```

$ less DM34740AaApep_vs_G0.blastp.annot.G0.gff
> /DM34740AaApep439

```

or to speed up the search by looking only at the beginning of each line (what the little `^` denotes)

```

> /^DM34740AaApep439

```

Search for the protein object DM34740AaApep439 by using the search facility of your viewer/editor (for less it is forward slash /).

```

DM34740AaApep439      ANNOT8R_GO      supported_by_sequence_similarity      1      139
1e-28      .      ID=DM34740AaApep439:GO:31110;

```

```

Dbxref=DB:uniprot:Q9VZS5,local:5519,G0:0005515,G0:0005488;Name=DM34740AaApep439:G0:3110;
Note=protein binding (stat:0.03);organism=H.melpomene;score=1e-28
DM34740AaApep439      ANNOT8R_G0      supported_by_sequence_similarity      1      139
1e-40      .      .      ID=DM34740AaApep439:G0:3111;
Dbxref=DB:uniprot:Q5UAR0,local:4026,G0:0003735,G0:0005198;Name=DM34740AaApep439:G0:3111;
Note=structural constituent of ribosome (stat:0.90);organism=H.melpomene;score=1e-40
DM34740AaApep439      ANNOT8R_G0      supported_by_sequence_similarity      1      139
1e-40      .      .      ID=DM34740AaApep439:G0:3112;
Dbxref=DB:uniprot:Q5UAR0,local:5619,G0:0005622,G0:0005622;Name=DM34740AaApep439:G0:3112;
Note=intracellular (stat:0.90);organism=H.melpomene;score=1e-40
DM34740AaApep439      ANNOT8R_G0      supported_by_sequence_similarity      1      139
1e-28      .      .      ID=DM34740AaApep439:G0:3113;
Dbxref=DB:uniprot:Q9VZS5,local:6933,G0:0007052,G0:0009987;Name=DM34740AaApep439:G0:3113;
Note=mitotic spindle organization (stat:0.07);organism=H.melpomene;score=1e-28
DM34740AaApep439      ANNOT8R_G0      supported_by_sequence_similarity      1      139
1e-40      .      .      ID=DM34740AaApep439:G0:3114;
Dbxref=DB:uniprot:Q5UAR0,local:6346,G0:0006412,G0:0008152;Name=DM34740AaApep439:G0:3114;
Note=translation (stat:0.90);organism=H.melpomene;score=1e-40
DM34740AaApep439      ANNOT8R_G0      supported_by_sequence_similarity      1      139
1e-40      .      .      ID=DM34740AaApep439:G0:3115;
Dbxref=DB:uniprot:Q5UAR0,local:5817,G0:0005840,G0:0005622;Name=DM34740AaApep439:G0:3115;
Note=ribosome (stat:0.90);organism=H.melpomene;score=1e-40
DM34740AaApep439      ANNOT8R_G0      supported_by_sequence_similarity      1      139
1e-28      .      .      ID=DM34740AaApep439:G0:3116;
Dbxref=DB:uniprot:Q9VZS5,local:1823,G0:0000022,G0:0009987;Name=DM34740AaApep439:G0:3116;
Note=mitotic spindle elongation (stat:0.07);organism=H.melpomene;score=1e-28
DM34740AaApep439      ANNOT8R_G0      supported_by_sequence_similarity      1      139
7e-39      .      .      ID=DM34740AaApep439:G0:3117;
Dbxref=DB:uniprot:Q962T2,local:15536,G0:0030529,G0:0005622;Name=DM34740AaApep439:G0:3117;
Note=ribonucleoprotein complex (stat:0.07);organism=H.melpomene;score=7e-39

```

You will notice that "protein binding" is in the Notes. You can use anything in the Note field to search your GBrowse. Try putting **intracellular** on the search box of the protein page.

This is pretty much all you need to know to use the database. If something is amiss, drop me a line at alexie -alpha symbol- butterflybase.org and I'll get back to you as soon as possible.

Happy gene hunting!

Using a LSF-driven PC-Farm

If you are doing many transcriptome projects, then computational power will be the limit, especially for annotation. It really helps if you have access to a PC-farm or a computing cluster. If you have a job management system based on LSF then you use the scripts I use for annotation. If you do have access to a computing cluster, I assume (and trust) that you fall in the **expert** user category so I'll keep this brief. Please note that the scripts are provided for the sake of your convenience and you will probably need to edit them a bit to customize it to your system. Hopefully will not spend as much time as if you wrote your own.

Filesystem Structure

We assume there is an NSF directory (e.g. your home) shared across all nodes. We also assume that each node has a scratch disk at /tmp that you can write to (about a couple of gigabytes will be needed, depending on what you're trying to BLAST).

You will need to copy all the files in the **lsf** directory to your home directory. The directory structure should be maintained as is but can be linked to any FileSystem you wish.

For example, we use a non-backed up FS for transfer, storage and tmp.

Databases should reside bzipped in the dbs/bz2 directory and a md5sum file at dbs/

you can use this command to create it from within "~/dbs/bz2"

```
$ md5sum *bz2 >../md5sums.txt
```

The FASTA directory contains some FASTA files which or may not be of use to your LSF approach (e.g. run SSAHA on a cluster)

BLASTs

The prepare_lsf.pl is used to prepare for blast runs. Run it from the "~/blasts/blast_in/" See the run_annot_afterparameterizing.sh for a bash file that will automatically generate all the required directories, files and launch the jobs if run from the above directory. The inputs of this file are just fasta files (also residing at "~/blasts/blast_in/").

Once the blasts are complete (or even before) you can use check_blasts_lsf.pl to see which have finished and relaunch them. There are some hard-coded values in this script to protect me from launching from execute nodes instead of the submit nodes (deimos): you will have to change them.

Because the scripts rely on the nodes having a /tmp directory where you can copy the databases (it really improves performance a lot...), one clever thing you can do is copy the databases to the nodes beforehand. Your system administrator will also love you as you will not have 100s of open files across the NSF.

InterProScan (IPRSCAN)

For this program you can use the run_iprscan.pl program. The iprscan has to be properly installed beforehand, including all the /data libraries that are shipped from EBI.

run_iprscan.pl is used like like prepare_lsf.pl but takes no options: the only arguments is a list of FASTA files (from within "~/blasts/blast_in/").

est2assembly citation

Alexie Papanicolaou, Remo Stierli, David G. Heckel and Richard ffrench-Constant: Next generation transcriptomes for next generation genomes using *est2assembly*. In preparation: please ask for an update

Remember: you **must** cite **all** of the programs used in the pipeline (see INSTALL file) and also Chado and GMOD. For software, it is an omission which is not uncommon but it is not very nice (and is academic misconduct). Typically, one includes in their methods: Data processed by [platform - *est2assembly* in this case] [ref] which uses prog1 [ref], prog2 [ref] etc. If you choose not to cite them properly, then the publication citation indexes of the software is not increased: i.e we get no public funding to provide the methods you just used for free to publish your data...

Acknowledgements

From our paper:

We would like to thank the following for making pre-publication data available: Chris Jiggins and his laboratory, Owen McMillan and his laboratory, Yannick Pauchet, Iva Fuková, Haobo Jiang and Heiko Vogel. Bastien Chevreux provided development versions of MIRA and excellent support, Jose Blanca provided *sff_extract*, James Wasmuth provided support for *prot4EST*, Ralf Schmid for *annot8r*, Derek Huntley for *SEAN*. David Clements and Scott Cain helped with *chado* and *Gbrowse*. We also thank the TU-Dresden Deimos PC-Farm for computational support. The authors report no conflicting interests. AP conceived, designed and performed the study; analysed and interpreted data; coded the software and drafted the manuscript. RS co-authored the GFF writing software and the *Gbrowse* schema. RHfC drafted the manuscript, financed and provided infrastructure for the study. All authors approved the final version of the manuscript. AP was supported by a Max Planck Stipendium by the Department of Entomology (MPI for Chemical Ecology) and the European Union Research Network GAMEXP.

DISCLAIMER & LICENSE

This software is released under the GNU General Public License version 3 (GPLv3).

It is provided "as is" without warranty of any kind. You can find the terms and conditions at <http://www.opensource.org/licenses/gpl-3.0.html>. Please note that incorporating the whole software or parts of its code in proprietary software is prohibited under the current license.