RUNNING IGGPIPE TO GENERATE IGG MARKERS

Version v0.0, Aug 2015

Table of Contents

This describes how to run the IGGPIPE software to analyze two or more genome sequences and produce a file of candidate IGG marker primer pairs for amplifying length-polymorphic regions of those genomes.

1. If not done already, download IGGPIPE files

1. Browse to https://github.com/BradyLab/IGGPIPE
2. At bottom of right column on screen, click "Download ZIP" and choose a place to put it on your computer.
3. Unzip the zip file on your computer.
4. Rename the unzipped folder from "IGGPIPE-master" to just "IGGPIPE".

2. If not done already, install IGGPIPE

- Look inside the downloaded IGGPIPE folder on your computer for file INSTALL.pdf or INSTALL.html and open either one and follow the instructions.

3. Requirements for running IGGPIPE

1. **Obtain genome FASTA files**

   IGGPIPE uses as input two or more FASTA files of complete genome sequences (at least one of which is assembled into chromosomes, the other(s) can be in the scaffold state). You should know ahead of time **which** genomes you are comparing. If you haven't already, download their genomic FASTA files, which are required.

2. **Work from the command line**

Most work here is done from the command line, by opening the Terminal application. Commands will be shown here and you may be able to get by with no knowledge of the command line, other than knowing how to start it (by starting the Terminal app on the Mac). However, ideally you should be familiar with some basic command line commands such as *cd, ls, cp, rm, mv, sudo, less, head* and *tail*, *man*, and *mkdir*. If you do not know how to use the command line or don't know these basic commands, you may want to take some time to learn a bit. Here are two short tutorial web sites:

http://www.davidbaumgold.com/tutorials/command-line

http://mac.appstorm.net/how-to/utilities-how-to/how-to-use-terminal-the-basics

Here is a longer tutorial from UC Davis:

http://korflab.ucdavis.edu/Unix_and_Perl/current.html

3. **Work in the IGGPIPE main directory unless otherwise instructed**

   While working from the command line to install IGGPIPE, most of the time you will be in the IGGPIPE main directory, unless instructed otherwise. If you unzipped the IGGPIPE zip file in your Documents folder, you would change into the IGGPIPE directory with this command:

   ```
   cd ~/Documents/IGGPIPE
   ```

4. **Know how to use a plain text editor and have one available**

   You must have a plain text editor you know how to use. If nothing else, the Mac "TextEdit" program will work (use Plain Text format). The free open-source TextWrangler program is strongly recommended, available from the Apple App Store (Applications, App Store) or from:

   http://www.barebones.com/products/textwrangler

4. Running IGGPIPE

1. **Assign a single capital letter name to each genome**

   IGGPIPE makes frequent use of a single capital letter to refer to a genome. For example, filenames and tab-separated file column names use such letters. Choose a single capital letter you will use to represent each of your genomes. For example, I used H=Heinz and P=pennellii for some of my testing.

2. **Copy allParameters.template file to new file allParameters.XY**

   IGGPIPE has a number of parameters that must be set to the values you desire. These are contained in plain text files whose name starts with "allParameters". IGGPIPE provides a template file named **allParameters.template** that you copy and edit to create your own "allParameters" files containing the parameter settings you want.

   You should make a new parameter file for each different set of settings you want to run with IGGPIPE. For example, each new set of genomes would have a different parameter file. Also, if you decided to make two different sets of markers from the same genomes, using two different

parameter settings, you might make two different parameter files. Here, we assume to begin with that you are only running your genomes with two genomes and a single set of parameters, and we will name the parameter file allParameters.XY, where X and Y are capital letters you chose to represent the two genomes. It is convenient to include those letters in the file name, to help you keep multiple parameter file straight.

Copy allParameters.template to allParameters.XY (substituting your genome letters for X and Y), using either the Mac's Finder or via the command line. For example:

```
cd ~/Documents/IGGPIPE    (or whatever is appropriate for your system)
cp allParameters.template allParameters.HP    (here, X=H, Y=P)
```

Besides the sample parameter file "allParameters.template", there are several more sample "allParameters" parameter files in the subfolder *allParameters*. These were used for testing IGGPIPE with various genomes. The subfolder *allParameters* was made to keep the files better organized and keep the root folder less messy. You may want to adopt the same strategy and put your "allParameters" files in the *allParameters* subfolder or in a subfolder you make yourself.

3. **Open the allParameters.XY file in your plain text editor**

   Open the new **allParameters.XY** file created above in your plain text editor for editing. For example, you might open the *nano* text editor from the command line:

   ```
   nano allParameters.XY
   ```

   If you are in a hurry, you don't need to read anything in the file, but can simply **search for "##\#"**, which are comment lines marking items that may need to be changed.

   Each "#@@" comment says whether it must be changed, might need to be changed, or probably will never need to be changed, etc. Many of these items typically will never need to be changed, so the actual number of changes that need to be made is smaller than it might first appear. It is recommended that rather than hurrying, you take time to read through the file, as the comments explain the purpose of each parameter, and you will want to know this information, at least for key parameters, to select the right parameter values for your needs.

4. **Search for "#@@" and set parameters to desired values**

   Search for "#@@" in the allParameters.XY file and check each one to see if it needs to be changed. If so, set it to the value you desire. Parameters you will definitely want to review and consider are:

   a. K

   b. N_GENOMES

   c. GENOME_NUMBERS

   d. GENOME_1, GENOME_2, etc.

   e. DIR_IGGPIPE_OUT

   f. PATH_GENOME_FASTA_1, PATH_GENOME_FASTA_2, etc.

   g. LMIN

h. DMAX

i. AMIN and AMAX

j. ADMIN and ADMAX

k. NDAMIN

l. OVERLAP_REMOVAL

m. EPCR_MAX_DEV

n. EPCR_MAX_MISMATCH and EPCR_MAX_GAPS

The next section gives guidance on choosing a value for K.

5. **Choosing a value for K**

The value of K must be chosen carefully. The larger the value, the more common unique k-mers will be found, up to a point, beyond which the number will fall because unique k-mers will begin to be long enough to no longer be in common with the other genome. The computational demands of IGGPIPE in the steps that immediately follow the search for common unique k-mers are directly proportional to the number of such k-mers, which argues for keeping K as small as possible while not so small that there won't be enough common unique k-mers for marker identification. We have successfully used 7 to 10 million common unique k-mers for producing markers; several hundred thousand might produce too few markers, while several tens of millions might create an intolerable computational demand.

The optimum value of K depends on the genetic architecture and amount of polymorphism between genomes. Very different genomes will have a peak number of common unique k-mers at a smaller K, so a smaller value would be a good choice. However, this does not necessarily mean that very similar genomes should use a larger value of K. Two *Arabidopsis thaliana* accessions will produce a much larger number of common unique k-mers for a given K, since most unique k-mers will be common. The challenge in that case is not to find enough of them, but to avoid having so many that computational resources are overwhelmed. We found that a value of 14 worked well for the quite different *S. lycopersicum* and *S. pennellii* genomes, but for the Arabidopsis accessions we tested, a value of 14 produced so many common unique k-mers that computation time was relatively long, while a value of 13 produced almost as many IGG markers in a much shorter time (see the table in the IGGPIPE paper that shows computation time for these two values of K in Arabidopsis accessions). Polyploid species might require a larger value of K to locate enough unique k-mers in the repeated genomes.

We advise checking the number of common unique k-mers obtained with a given value of K the first time the pipeline is run on a given set of genomes, and adjust K if necessary. To speed this process, the pipeline can be run only through the stage where the common unique k-mers are produced, as described below. If too few k-mers result, then the user should increase K by one, and if too many k-mers result, the user should decrease K by one. The pipeline is then run to completion to produce the IGG marker primer sequences, and the total number of primer pairs produced is examined. If there are too few, it is advisable to rerun the entire pipeline with K both increased and decreased by one.

6. **Additional notes on setting parameter values**

   The parameter DIR_IGGPIPE_OUT is the path of the folder where all output will be placed. You don't need to change this parameter but you should note what its value will be, and change it if you want to.

   The parameters PATH_GENOME_FASTA_1, PATH_GENOME_FASTA_2, and so on, up to the number of genomes being analyzed, are the paths to the FASTA files for the genomes to be analyzed. If any of these contain scaffolds, you may want to consider whether you should remove smaller scaffolds. The e-PCR portion of the pipeline takes an inordinately long amount of time to run when there are tens to hundreds of thousands of scaffolds. If this is your case, and if many of the scaffolds are very small and not likely to contribute to viable markers, you should remove them from the FASTA file to be analyzed.

   After finishing changes, save the modified allParameters.XY file.

7. **Check Primer3 settings in primer3settings.txt if desired**

   The file **primer3settings.txt** contains parameter settings for Primer3, which is used to generate the actual primers. It is possible that you might want to use different Primer3 settings from the defaults listed in this file. If so, edit the file and make the desired changes. For example, you might change the parameters that determine the acceptable *range of primer Tm values*. If you have several different setting values you use, you will probably want to keep a directory of different primer3settings.txt files and copy the needed one to "primer3settings.txt" prior to each run of IGGPIPE. When testing IGGPIPE using the allParameters.test file that was created during installation, always copy primer3settings.default.txt to primer3settings.txt first.

   The Primer3 user manual (http://primer3.sourceforge.net/primer3_manual.htm) describes all the Primer3 settings file parameters.

8. **Understand use of *make* and "Makefile" for running IGGPIPE**

   The IGGPIPE software consists of multiple software applications that progressively analyze the genome sequence data and eventually produce candidate IGG marker primers. The task of running all this software has been automated using a "Makefile", which is a file with that name containing commands formatted correctly for reading the allParameters.XY parameter file and running the software applications. The Makefile is applied by using the application named *make*, which was installed when IGGPIPE was installed, if it didn't already exist.

   A big advantage of using "Makefile" and *make* is that if something goes wrong (and by Murphy's law, it probably will), the portion of the work successfully completed is not lost, and does not need to be repeated. This is important because it can take quite a long time to run genomes all the way through the IGGPIPE software. Depending on your computer speed and memory, it can take hours or even days. If an error occurs, *make* will stop, and an error message should be visible on the terminal. After fixing the error, all you have to do resume the pipeline commands from the last successful step is re-enter the same *make* command. *make* knows which step to start at because it knows all the files to be produced by the pipeline, as they are specified in allParameters.XY, and it checks to see if the files exist, and starts at the pipeline step whose output file does not exist. Manually deleting a file will also cause *make* to run the pipeline starting at the step needed to make that file.

You must finish editing the allParameters.XY file before trying to run the pipeline using *make*. If that file is ready to go, you can start running IGGPIPE using the command *make* from the command line, with additional command arguments. The first argument that is required is of the form "PARAMS=<allParameters filename>". For example, if your allParameters file is named "allParameters.XY", then the *make* command starts out as *make PARAMS=allParameters.XY*.

The remaining command arguments for the *make* command tell which part of the pipeline to run. If no additional argument is given or if the argument is *ALL*, the entire pipeline is run (or as much of it as is needed to resume where a previous error had halted). However, since the choice of some of the parameters, especially the value of K, can have a strong influence on the number of markers found, it is best to run IGGPIPE a few steps at a time and check the output after those steps before proceeding further. The following sections will guide you in this.

Use this command to get a listing of complete usage information for running *make*:

```
make usage
```

That command will use the *less* command to display file *help.txt*. Press the space bar to move through the text, or press *q* to exit from the help text.

If at any point you want to remove ALL files already generated and start anew, you can do that with this command:

```
make PARAMS=allParameters.XY CLEAN=1 ALL    (replacing with your allParameters name)
```

Running IGGPIPE with a *make* command will usually produce a lot of output on the terminal, and some of this output may be important to examine, especially if an error occurs. Since the output might scroll off the screen and be unavailable, it is a good idea to save it, and this can be done by using the *tee* command along with the *make* command. The *tee* command can write everything that is displayed on the terminal to a file also. You might want to make a folder to contain these "log" files:

```
mkdir logFiles
```

To use *tee*, choose a log file name, let's say *make_HP14.txt*, and then add at the end of your *make* command line the extra commands | *tee logFiles/make_HP14.txt*, as in this example:

```
make PARAMS=allParameters.XY ALL | tee logFiles/make_HP14.txt
```

Then, after *make* finishes, you can examine that log file at any time to see what the pipeline output was, for example:

```
less logFiles/make_HP14.txt
```

You should use the *tee* command each time you run the pipeline unless you are sure you won't want to reexamine the output later. We will not show the *tee* command in the instructions below, however. It is up to you to decide whether to use it.

We have run IGGPIPE on several different genomes to try to anticipate unusual problems and handle them without error, but there are probably many situations that we haven't yet encountered. If you email us with information about errors and their resolution if you were able to resolve

them, we'll try to make improvements to IGGPIPE in error handling and in its input data format flexibility to help future users that encounter the error.

9. **Run IGGPIPE with the command "make PARAMS=allParameters.XY getGenomicPosIsect GENOME=1"**

   The first several steps in the pipeline extract unique k-mers from the FASTA files of the genomes, intersect these to produce a list of common unique k-mers, and add genomic positions to them for each of the genomes. To run these steps, use this command:

   ```
   cd ~/Documents/IGGPIPE    (or whatever is appropriate for your system)
   make PARAMS=allParameters.XY getGenomicPosIsect GENOME=1    (replacing with your allPa
   ```

   or, better yet, log to a file also:

   ```
   make PARAMS=allParameters.XY getGenomicPosIsect GENOME=1 | \
   tee logFiles/make_getGenomicPosIsect_HP14.txt
   ```

   If it completes successfully, the end of the command output will show the message:

   ```
   getGenomicPosIsect file(s) for genome(s) 1 are up to date.
   ```

   If it says something else, indicating an error occurred, examine the output carefully and try to diagnose and fix the error, then enter the above *make* command again to retry the failed step.

   Assuming the first steps completed successfully, count the number of k-mers in the common unique k-mer file for genome 1, which is located in the *Kmers* subfolder of the output folder you specified in your allParameters file for parameter DIR_IGGPIPE_OUT. The name of the file is *Kmers_1.isect* and it is a text file containing one k-mer per line, with the position information for that k-mer following the k-mer on the line. You can look at it with the *less* command if you want to. To count the number of k-mers in it, use the *wc -l* command, which counts lines in a file, like this (replacing *outFolderForMyProject* with your output folder name):

   ```
   wc -l outFolderForMyProject/Kmers/Kmers_1.isect
   ```

   If it shows that you have, say, several million or more, that is good. A few million or less might be too few to generate enough markers. Tens of millions might be too many and cause subsequent pipeline steps to take a very long time. The number of k-mers is influenced by both the value of K in the parameter file and by how different the genomes are. Very similar genomes might never have several millions of k-mers. The number of common unique k-mers increases as K is increased, up to a point, then starts to decrease. For similar genomes, you might decide you have too many k-mers and should edit the parameter file and decrease K by 1 to keep computation time reasonable. Or, you may wish to try running the rest of the pipeline with tens of millions of common unique k-mers and check to see how long the computation time actually is. With the tomato/pennellii genomes, K=14 was a good number. With *Arabidopsis thaliana* Col-0 and Ler-0 accessions, which were much more similar to one another than tomato/pennellii, a value of K=13 worked better, because K=14 produced so many k-mers that computation time was excessive (half an hour vs three hours). For very large genomes, you might need to increase K by 1. Since the number of k-mers goes up dramatically with increasing K, you probably will never raise K above 15 or perhaps 16.

The default setting in the parameter file for the output directory parameter DIR_IGGPIPE_OUT is to include the value of K in the directory name. This means you can run IGGPIPE with one value of K, then change it and run it again and the output will go into a new directory.

If you feel you have too few or too many k-mers, then you should increase or decrease K by 1 and try again. Try both an increase and a decrease in K to see how the number of k-mers is affected. Set the parameter file to the value of K that seems best to you for proceeding with additional pipeline steps.

10. **Run IGGPIPE with the command "make PARAMS=allParameters.XY findLCRs"**

The next few steps of the pipeline analyze the common unique k-mers to find locally conserved regions (LCRs). To run these steps, use this command:

```
make PARAMS=allParameters.XY findLCRs    (replacing with your allParameters name)
```

If it completes successfully, the end of the command output will show the message:

```
findLCRs files are up to date.
```

If it says something else, indicating an error occurred, examine the output carefully and try to diagnose and fix the error, then enter the above *make* command again to retry the failed step.

Assuming the steps completed successfully, the LCRs output file will now be located in the main output directory (set with the DIR_IGGPIPE_OUT parameter). Its name starts with *LCRs_*, and with many command line interfaces you don't need to enter the full name in a command, but can instead enter *LCRs_* and then press the tab key to auto-complete the remainder of the file name. In the example code below, we will show the LCRs_ filename as "LCRs_*.tsv", and will use the "*" character in other filenames below to stand in for the long character name.

You can look at the first few lines of the LCRs file with the *head* command, or you can load the file into Excel or a text editor to examine it, if you want. You should definitely count the number of LCRs in the file. The file has one common unique k-mer per line, and the k-mer belongs to one LCR only, whose ID is given in the last column of the file. Thus, to count the number of LCRs, you need to count the number of unique values in the last column. You therefore need to know the column number of the last column. The number of columns in the LCRs file is 2+5*N_GENOMES, since the file contains five columns of data for each genome that is processed, plus two additional columns (k-mer is first column, LCR ID is last column). For the usual case of two genomes, the last column is column 12, and you can count the number of LCRs with this command:

```
cut -f 12 outFolderForMyProject/LCRs*.tsv | uniq | wc -l
```

This command line is also included in shell file code/countLCRsInLCRfile.sh, which automatically computes the last column number and counts LCRs, given two arguments: the LCRs file name and the value of N_GENOMES (number of genomes). For example:

```
source code/shell/countLCRsInLCRfile.sh outFolderForMyProject/LCRs*.tsv 2
```

If you had too few common k-mers you might also have too few LCRs. A million or more LCRs would be nice. The fewer you have, the fewer markers you are likely to get. If there seem to be too few, check the pipeline output. It will show the number of common unique k-mers it processes (it

processes them in batches), and the number remaining after it enforces DMIN, LMIN, and KMIN on the reference genome. If these numbers fall dramatically towards 0, it indicates that either there are no good LCRs between the two genomes, or the parameters DMIN, LMIN, and/or KMIN might be too strict. However, expect a pretty big drop with the LMIN step, because typically a large fraction of the common unique k-mers are too close together, with too much separation from the next k-mer, to form a useful LCR.

If you feel you have too few LCRs, then you should try editing the parameter file and changing the DMIN, LMIN, and/or KMIN parameters and try again. The default value for the LCRs_ filename, set by the parameters SFX_LCR_FILE and PATH_LCR_FILE, includes the values of DMIN, LMIN, and KMIN in the filename, so if you change the values, you can simply re-run the pipeline with the same *make* command, and it will generate a new LCRs_ file with a different name, without repeating preceding pipeline steps that do not need to be repeated.

Set the parameter file to the values for DMIN, LMIN, and KMIN that seem best to you and re-run this pipeline step before proceeding with additional pipeline steps.

The columns in the LCRs* file are described in Table 4.

11. **Run IGGPIPE with the command "make PARAMS=allParameters.XY findIndelGroups"**

The next step of the pipeline analyzes the LCRs to find Indel Groups that satisfy the parameters AMIN, AMAX, ADMIN, ADMAX, NDAMIN, and MINFLANK. To run this step, use this command:

```
make PARAMS=allParameters.XY findIndelGroups    (replacing with your allParameters nam
```

If it completes successfully, the end of the command output will show the message:

```
findIndelGroups files are up to date.
```

If it says something else, indicating an error occurred, then as usual, examine the output carefully and try to diagnose and fix the error, then enter the above *make* command again to retry the failed step.

Assuming the step completed successfully, count the number of Indel Groups in the two output files. One output file includes all Indel Groups found, even when they overlap one another. The other output file includes only non-overlapping Indel Groups, which were determined based on the setting of the parameter OVERLAP_REMOVAL. The output files are located in the main output directory and their names start with *IndelGroups* followed by *Overlapping_* and *Nonoverlapping_*. You can look at the first few lines with the *head* command or load the files into Excel or a text editor to examine them, if you want. The command line to count the Indel Groups in both files would look like this:

```
wc -l outFolderForMyProject/IndelGroups*
```

The number of Indel Groups is of the same order of magnitude as the number of markers you will obtain, so pay close attention to the count. If you had too few LCRs you might also have too few markers. Tens of thousands of overlapping markers and thousands of non-overlapping ones are nice numbers to have. If there seem to be too few, you may want to experiment with different values for the AMIN, AMAX, ADMIN, ADMAX, NDAMIN, and MINFLANK parameters. As with the LCRs, these values are normally part of the IndelGroups output file filenames, so if you

edit the parameter file and change the values, then re-run the pipeline by entering the same *make* command above, new files are produced with new names.

Set the parameter file to the values for the parameters that seem best to you and re- run this pipeline step before proceeding with remaining pipeline steps.

The columns in the IndelGroups* files are described in Table 3.

12. **Run IGGPIPE with the command "make PARAMS=allParameters.XY findPrimers"**

The next several steps of the pipeline extract DNA sequences for each Indel Group from all genomes using the parameter EXTENSION_LEN, then Primer3 (actually, primer3_core) is run to design primers for each Indel Group, using the primer design parameters in file primer3settings.txt.

To run these steps, use this command:

```
make PARAMS=allParameters.XY findPrimers    (replacing with your allParameters name)
```

Depending on the number of Indel Groups, this can take a *long* time. We have seen it take two days to finish running all Indel Groups through Primer3.

If it completes successfully, the end of the command output will show the message:

```
findPrimers files are up to date.
```

If it says something else, try to diagnose and fix the error as usual, then enter the same *make* command again to retry the failed step.

Assuming the steps completed successfully, count the number of IGG marker primer pairs in the output file, which is located in the main output directory and has a name that starts with *NonvalidatedMarkers_*. You can look at the first few lines with the *head* command or load the files into Excel or a text editor to examine them, if you want. The command line to count the primer pairs in the file would look like this:

```
wc -l outFolderForMyProject/NonvalidatedMarkers*
```

Each line of the file contains one pair of primers. Each pair is a candidate IGG marker, but they have not yet been validated using e-PCR, which will mark a few of them as bad and remove them.

The columns in the NonvalidatedMarkers* file are described in Table 1.

13. **Run IGGPIPE with the command "make PARAMS=allParameters.XY removeBadMarkers"**

The next several steps of the pipeline run e-PCR on each primer pair in the NonvalidatedMarkers* file. All markers are run through e-PCR once for each genome. After that, an R script is run which examines the e-PCR results and removes from the NonvalidatedMarkers* primer pairs all those pairs that failed the e-PCR test in or more genomes, and writes new files with the validated IGG marker primer pairs.

To run these steps, use this command:

```
make PARAMS=allParameters.XY removeBadMarkers    (replacing with your allParameters na
```

Depending on the number of primer pairs, this can take a *long* time. We have seen it take two days *per genome* to finish running all primer pairs through e-PCR.

If it completes successfully, the end of the command output will show the message:

```
removeBadMarkers files are up to date.
```

If it says something else, try to diagnose and fix the error as usual, then enter the same *make* command again to retry the failed step.

Assuming the steps completed successfully, count the number of validate IGG primer pairs in the two output files. One output file includes all validated primer pairs, even when their amplicons overlap one another. The other output file includes only primer pairs that produce non-overlapping amplicons, determined based on the setting of the parameter OVERLAP_REMOVAL. The output files are located in the main output directory and their names start with *Markers* followed by *Overlapping_* and *Nonoverlapping_*. You can look at the first few lines with the *head* command or load the files into Excel or a text editor to examine them, if you want. The command line to count the primer pairs in both files would look like this:

```
wc -l outFolderForMyProject/Markers*
```

Each line of the file contains one pair of primers. Each pair is an e-PCR-validated IGG marker, essentially the final output of the pipeline.

If you are running IGGPIPE with the NDAMIN parameter set greater than 2, you might wish to count the number of markers with NDA=2, NDA=3, etc., to see how many markers there are with different numbers of distinct amplicons. The second column of the marker files is the NDA column, and this command will search for all such lines with a 2 in that column and count them:

```
cut -f 2 outFolderForMyProject/Markers* | grep -E "^2$" | wc -l
```

The shell file code/countMarkersInMarkerFile.sh is provided to automate this for all values of NDA. It counts markers, given two arguments: the Markers file name and the value of NDAMIN. It requires a single file name, and won't work if wildcards are used to select multiple files. For example:

```
source code/shell/countMarkersInMarkerFile.sh outFolderForMyProject/MarkersOverlapping
```

The columns in the Markers* files are described in Table 1.

14. **Run IGGPIPE with the command "make PARAMS=allParameters.XY ALL"**

Now you can run the rest of the pipeline to finish up. The final step produces some plots of marker statistics and density. This is done with this command:

```
make PARAMS=allParameters.XY plotMarkers    (replacing with your allParameters name)
```

However, you can also use this command, which is the command that runs the entire pipeline, all steps from start to finish:

```
make PARAMS=allParameters.XY ALL    (replacing with your allParameters name)
```

This same command could have been used from the very start, to simply run the entire pipeline. It can be used *at any time* to re-run the pipeline. For each step of the pipeline that was previously run successfully, it will simply output a message saying that the files from that pipeline step are up to date. However, for any step that was either not run at all, or failed, it will attempt to re-run that step and following steps that depend on it. At the very end, when it has successfully completed all pipeline steps, it issues this message:

```
ALL files are up to date
```

This is the message you want to see to know that the pipeline has completed successfully.

If you edit the parameter file and change parameters (so that the names of the files that are produced are also changed, since the parameter values are contained within the file names), then when you use the above *make* command, the pipeline re-runs starting at whatever step uses the changed parameters, so you can easily make parameter changes and try again.

Sometimes you may want to force the pipeline to re-run starting at a certain step. For example, maybe you want to re-run the pipeline starting at the *findLCRs* step. To do this, you can either delete the output files produced by that step, or use this command to delete them:

```
make PARAMS=allParameters.XY CLEAN=1 findLCRs    (replacing 'findLCRs' with whatever s
```

After that, you can re-run the pipeline with the *make … ALL* command shown above to generate the files anew.

The final step, plotMarkers, produces several output files in the main output directory. One file is a .pdf file whose name starts with MarkerCounts_, containing plots of counts of markers on each chromosome of each genome. You can examine that file with any .pdf file viewer. The other files are .png image files whose names start with MarkerDensity_ and end with *_X.plot.png*, where X is replaced with the genome letters you assigned for your analysis. Each .png file has an image of the chromosomes with lines showing the positions of each marker. When multiple scaffolds are used, the file limits the output to the first several scaffolds.

15. **Comments about pipeline result files**

Here is a summary of the filenames produced by each *make* step, in pipeline order:

***make* command or other command**	**Produces output file(s)**
a. make PARAMS=myFilename getSeqInfo	GenomeData/*.idlens
b. make PARAMS=myFilename getContigFile	GenomeData/*.contigs
c. make PARAMS=myFilename getKmers	Kmers/Kmers_*.kmers
d. make PARAMS=myFilename kmerStats	Kmers/Kmers_*.stats
e. make PARAMS=myFilename kmersToText	Kmers/Kmers_*.kmers.txt
f. make PARAMS=myFilename getGenomicPosIsect	Kmers/Kmers_*.isect
g. make PARAMS=myFilename mergeKmers	Kmers/Kmers_*.merge
h. make PARAMS=myFilename sortCommonUniqueKmers	Kmers/common.unique.kmers

***make* command or other command**	**Produces output file(s)**
i. make PARAMS=myFilename findLCRs	LCRs_*.tsv, BadKmers_*.tsv
j. make PARAMS=myFilename findIndelGroups	IndelGroupsOverlapping_*.tsv, IndelGroupsNonoverlapping_*.tsv
k. make PARAMS=myFilename getDNAseqsForPrimers	IndelGroupsOverlapping_*.dnaseqs
l. make PARAMS=myFilename findPrimers	NonvalidatedMarkers_*.tsv
m. make PARAMS=myFilename ePCRtesting	MarkerErrors_*.tsv
n. make PARAMS=myFilename removeBadMarkers	MarkersOverlapping_*.tsv, MarkersNonoverlapping_*.tsv
o. make PARAMS=myFilename plotMarkers	MarkerCounts_*.pdf, MarkerDensity_*.png
p. make PARAMS=myFilename getDNAseqsForIndelsSNPs	*.withseqs.tsv
q. make PARAMS=myFilename IndelsSNPs	*.indels.tsv, *.snps.tsv
r. make PARAMS=myFilename plotIndels	*.indels.pdf
s. Rscript code/R/dotplot.R dotplot.template	LCRs_*.dotplot.png
t. Rscript code/R/annotateFile.R annotate.template	MarkersAnnotated_*.tsv
u. Rscript code/R/annotateFile.R annotate/HP11_isInNearColumn.markers	MarkersAnnotated_WithInNearFeatures_*.indels.tsv
v. Rscript code/R/annotateFile.R annotate/HP11_to_gff3.markers	MarkersAnnotated_GFF3_*.gff3

(Note that some of the files listed above are produced by steps to be described below).

The marker file names and other file names of files in the main output directory are very long and cumbersome, because they include parameter values in them. You may want to copy files to a shorter name to work with them.

The meaning of *overlapping* and *non-overlapping* should be clear from the explanation of the parameter OVERLAP_REMOVAL in the comments in allParameters.XY.

The various .tsv files can be loaded into Excel to examine, and they can also be post-processed (see below) to change them into other formats.

Tables describing each column in each .tsv file type are at the end of this document.

5. Post-processing tools

1. **Finding Indels and SNPs**

   Pipeline software is also provided to read a file of LCRs, non-overlapping Indel Groups, or non-overlapping Markers, extract the DNA sequences from the genomes in each LCR or Marker region and align them, then locate all Indels and SNPs in the aligned sequences and write their positions to files. This part of the pipeline is NOT run when the *make … ALL* target is built. To

run this and find Indels and SNPs, use *make … IndelsSNPs*, after setting the parameters in your "allParameters" parameter file. These are PATH_INDELS_SNPS_INPUT_FILE (the input file name), MAX_INDELS_PER_KBP and MAX_SNPS_PER_KBP (maximum number of Indels and SNPs that may occur in an alignment per Kbp of sequence, and if exceeded, cause the alignment to be discarded as unreliable), and SCRAMBLE_SEQUENCE (can be set TRUE to do alignments with scrambled sequences, to determine the actual numbers of Indels and SNPs per Kbp in random sequence alignments. Run the Indel/SNP finder as follows:

```
make PARAMS=allParameters.XY IndelsSNPs    (replacing with your allParameters name)
```

This produces two files in your output folder whose names end with "indels.tsv" and ".snps.tsv", containing tables of all Indels and SNPs found in either the LCR or Indel Group or marker regions (depending on the setting of PATH_INDELS_SNPS_INPUT_FILE). Examine them to see the data they contain. The columns are described in Table 5 and Table 6.

The default settings for the parameters MAX_INDELS_PER_KBP and MAX_SNPS_PER_KBP are reasonable values, but you can adjust them to either decrease FALSE positives at the expense of fewer TRUE positives (smaller values) or increase TRUE positives at the expense of more FALSE positives (larger values).

Note that the pipeline for *make IndelsSNPs* automatically invokes a preceding step, *make getDNAseqsForIndelsSNPs* that extracts DNA sequences in preparation for alignment, and writes them to a file whose name ends with "withseqs.tsv".

2. **Plotting Indel information**

   Another R program that is NOT run as part of the pipeline when the *make … ALL* target is built, but which can be run using *make … plotIndels*, reads the Indels file produced by *make … IndelsSNPs* and plots information from it in a pdf file. The program is called plotIndels.R. Run it as follows:

   ```
   make PARAMS=allParameters.XY plotIndels    (replacing with your allParameters name)
   ```

   This produces a file in your output folder whose name ends with "indels.pdf", containing plots of various Indel information. Examine it to see the plots it contains.

3. **Dot plots**

   The LCRs_*.tsv output file contains locally conserved regions associated with common unique k-mers. It represents a whole genome alignment between the genomes used in IGGPIPE analysis. An R program, dotplot.R, is provided that can plot this data as a dot plot.

   This program is run by first copying the text file "dotplot.template" to a new name (e.g. dotplot.XY) and editing it to specify the parameters of the dot plot. Comments in the file describe each parameter. The program is then run from the command line with a command like this:

   ```
   cd ~/Documents/IGGPIPE    (or whatever is appropriate for your system)
   Rscript code/R/dotplot.R dotplot.XY    (or whatever name you gave the parameter file)
   ```

   When it finishes running, the dot plot output file can be found in the place and under the name specified in the parameter file. Use multiple parameter files with different settings to explore different regions of the genomes in greater resolution (parameters include what region of the genome is to be plotted).

The "dotplot.template" file is configured for generating a dot plot file using the LCRs generated via the allParameters.test configuration file.

Besides the sample parameter file "dotplot.template" (which has settings for testing the IGGPIPE installation), there are several more sample "dotplot" parameter files in the subfolder *dotplot*, that were used for plotting data from various genomes that IGGPIPE was tested with, although the parameter file is straightforward and you probably don't need other examples to work from. You may want to put your own "dotplot" parameter files in subfolder *dotplot* or your own subfolder to keep them organized.

4. **Annotating marker files with other position data and producing GFF3 and GTF files**

You may want to make your marker data more conveniently available. For example, you might want to convert it to GFF3 file format so you can add a "marker" track to a genome browser. Or, you may have other genome position data that you would like to have associated with your marker data, such as a file giving positions of introgressions of one genome within another (you might want a column in the marker file showing which introgressions the marker was near). As another example, you might want to add a column in the marker file containing the names of the genes closest to the marker, and the distance to the genes. All of these situations and more can be handled by an R program, annotateMarkers.R, provided with IGGPIPE. The program can read and write files of type .tsv (tab-separated variable), .csv (comma-separated variable), .gff3 (general feature format), or .gtf (gene transfer format), all common formats used to hold genome browser track data or FASTA file annotation data. It can add, remove, edit, and rename columns. It can read two separate files and merge their data. It can convert from one of these file formats to another.

This program is run by first copying the text file "annotate.template" to a new name (e.g. annotateIntrogressions.XY or addGeneInfo.XY or makeGFF3.XY) and then editing it to specify the parameters for the annotation and/or file conversion. Comments in the file describe each parameter. The program is then run from the command line with a command like this:

```
cd ~/Documents/IGGPIPE    (or whatever is appropriate for your system)
Rscript code/R/annotate.R addGenes.XY    (or whatever name you gave the parameter file
```

When it finishes running, the output files can be found in the place(s) and under the name(s) specified in the parameter file.

Besides the sample parameter file "annotate.template" (which has settings for testing the IGGPIPE installation), there are several more sample "annotate" parameter files in the subfolder *annotate*, with file names hinting at what they do, and comments at the start of each file describing what it does. It may be easier to copy one of these and modify it for your needs. You may want to put your own "annotate" parameter files in subfolder *annotate* or your own subfolder to keep them organized.

So, the idea is to use multiple parameter files with different settings to do different types of annotation and file conversion.

Some of the sample parameter files generate .gff3 files that can be added as a track to a genome browser, to display markers in the browser. Instructions for adding the track are given in comments at the start of the parameter file. Two marker files, one for *Arabidopsis thaliana* Col-0 vs. Ler-0 accessions, and the other for *Solanum lycopersicum* vs. *Solanum pennellii* genomes, were created

to test IGGPIPE, and the marker files were converted to .gff3 files suitable for making a browser track. These files can be found in subfolders of the *annotate* folder.

File formats can be finicky, especially .gff3 files. An incorrectly formatted file will cause problems with annotateFile.R. When you have problems, if you can submit an issue to the GitHub repository named "BradyLab/IGGPIPE", and attach or insert a copy of your parameter file, that would be helpful. A copy of the input data files would probably also be needed to debug problems, but GitHub does not allow files to be attached. You can email them to us, or find some other way to send them.

5.1. For problems and help:

- Post an issue on GitHub under BradyLab/IGGPIPE repository
- Contact: Ted Toal, twtoal@ucdavis.edu [mailto:twtoal@ucdavis.edu]

6. Tables

Table 1. Columns in MarkersOverlapping_, MarkersNonoverlapping_, NonvalidatedMarkers_, MarkerErrors_ files; X,Y=chosen genome letters

Column	Description
NDA	Number of distinct amplicon sizes, in range NDAMIN..N_GENOMES
Xid	Genome X sequence ID
Xpct	Genome X percent of sequence ID length at which marker is located
XampLen	Genome X amplicon length
Yid	Genome Y sequence ID
Ypct	Genome Y percent of sequence ID length at which marker is located
YampLen	Genome Y amplicon length
YXdif	Difference in length between genomes X and Y amplicons, negative if genome X longer than genome Y
YXphase	Phase of amplicons between genomes X and Y, "+" if both amplicons run in same direction, "-" if opposite directions
prmSeqL	Left side or upstream primer sequence
prmSeqR	Right side or downstream primer sequence
prmTmL	Left side primer Tm
prmTmR	Right side primer Tm
prmLenL	Left side primer length
prmLenR	Right side primer length
XampPos1	Genome X amplicon starting (upstream) position
XampPos2	Genome X amplicon ending (downstream) position, XampPos2 always > XampPos1
YampPos1	Genome Y amplicon starting (upstream) position

Column	Description
YampPos2	Genome Y amplicon ending (downstream) position, YampPos2 > YampPos1 if YXphase is "+", < if "-"
kmer1	Common unique k-mer for left side primer region, canonical (exically smaller of k-mer and its reverse complement)
kmer1strands	N_GENOMES `"" and "-" characters for genomes 1..N_GENOMES. A ""` means k-mer 1 lies on the "+" strand in that genome, "-" means "-" strand.
kmer1offset	Offset in bp of outside (away from amplicon) edge of k-mer 1 from that end of the amplicon. A value of 0 means the amplicon and k-mer ends correspond, >0 means k-mer starts inside the amplicon, <0 means k-mers starts outside it.
kmer2	Common unique k-mer for right side primer region, canonical (exically smaller of k-mer and its reverse complement)
kmer2strands	Like kmer1strands, for k-mer 2.
kmer2offset	Like kmer1offset, for k-mer 2.
Xseq1	Genome X DNA sequence around left side primer region
Xseq2	Genome X DNA sequence around right side primer region
Yseq1	Genome Y DNA sequence around left side primer region
Yseq2	Genome Y DNA sequence around right side primer region

Table 2. Column reasonDiscarded in MarkerErrors_ files (see Table 1 for other columns)

reasonDiscarded	Description
found multiple	ePCR found multiple amplicons (expected reason)
not found	ePCR didn’t find amplicon (should never happen)
wrong seq id	ePCR sequence ID output is wrong (should never happen)
wrong pos	ePCR left and right position output is wrong (should never happen)
wrong posL	ePCR left position output is wrong (should never happen)
wrong posR	ePCR right position output is wrong (should never happen)

Table 3. Columns in IndelGroupsOverlapping_ and IndelGroupsNonoverlapping_ files; X,Y=chosen genome letters

Column	Description
kmer1	Common unique k-mer for left side primer region, canonical (lexically smaller of k-mer and its reverse complement)
kmer2	Common unique k-mer for right side primer region, canonical (exically smaller of k-mer and its reverse complement)
NDA	Number of distinct amplicon sizes, in range NDAMIN..N_GENOMES
Xid	Genome X sequence ID
Xpos1	Genome X position of upstream end of k-mer 1 on "+" strand

Column	Description
Xpos2	Genome X position of upstream end of k-mer 2 on "+" strand, Xpos1 < Xpos2 always
Xs1	Genome X k-mer 1 strand, "+" or "-"
Xs2	Genome X k-mer 2 strand, "+" or "-"
Xctg1	Genome X contig number within sequence Xid of contig containing k-mer 1
Xctg2	Likewise for k-mer 2, Xctg1 = Xctg2 always
XkkLen	Genome X distance from 5' end of k-mer 1 on "" strand to 5' end of k-mer 1 on "" strand
Xpct	Genome X percent of sequence ID length at which marker is located
Yid	Genome Y sequence ID
Ypos1	Genome Y position of upstream end of k-mer 1 on "+" strand
Ypos2	Genome Y position of upstream end of k-mer 2 on "+" strand, Ypos1 < Ypos2 if amplicon in X and Y genomes run in the same direction, > if opposite directions
Ys1	Genome Y k-mer 1 strand, "+" or "-"
Ys2	Genome Y k-mer 2 strand, "+" or "-"
Yctg1	Genome Y contig number within sequence Yid of contig containing k-mer 1
Yctg2	Likewise for k-mer 2, Yctg1 = Yctg2 always
YkkLen	Genome Y distance from 5' end of k-mer 1 on "" strand to 5' end of k-mer 1 on "" strand
Ypct	Genome Y percent of sequence ID length at which marker is located

Table 4. Columns in LCRs_ and BadKmers_ files; X,Y=chosen genome letters

Column	Description
(none, row name)	Common unique k-mer, canonical representation (the lexically smaller of k-mer and its reverse complement)
X.seqID	Genome X sequence ID
X.pos	Genome X position of upstream end of k-mer on "+" strand relative to start of X.seqID
X.strand	Genome X k-mer strand, "+" or "-"
X.contig	Genome X contig number within sequence X.seqID sequence of contig containing the k-mer
X.contigPos	Genome X position of upstream end of k-mer on "+" strand relative to start of X.contig
Y.seqID	Genome Y sequence ID
Y.pos	Genome Y position of upstream end of k-mer on "+" strand relative to start of Y.seqID
Y.strand	Genome Y k-mer strand, "+" or "-"
Y.contig	Genome Y contig number within sequence X.seqID sequence of contig containing the k-mer
Y.contigPos	Genome Y position of upstream end of k-mer on "+" strand relative to start of Y.contig

Column	Description
LCR	Integer LCR number to which this k-mer is assigned, each LCR has a unique LCR number assigned to it

Table 5. Columns in *.indels.tsv files; X,Y=chosen genome letters

Column	Description
ID	Unique ID tying row back to originating input file row. LCR input files: LCRnumber. IndelGroup and Markers files: refID_refPos1_refPos2.
phases	Phase of each genome incl. ref. genome, relative to ref. genome, string of +/- chars, + : same direction, - : opposite direction.",
idx	Starts at 1 and counts each Indel within an ID. For given ID (input row), number of Indels in that region is max idx value. If more than two genomes, entire region where alignment has a gap in one or more genomes is counted as one Indel even if multiple gap regions occur in different genomes.
Xdel,Ydel	Total number of deleted bps within the Indel in genomes X,Y. With 2 genomes, del = 0 in genome with insertion (no gaps), del > 0 in genome with deletion (gaps). With >2 genomes, del can be non-zero for all genomes. A genome has only insertions in the Indel if del is 0, and it has only deletions if end-start-1 = 0, and otherwise it has a mixture of at least one insertion and one deletion within the Indel interval.
Xid,Yid	Sequence ID of the Indel in genomes X,Y.
Xstart,Xend,Ystart,Yend	Overall Indel starting and ending position in genomes X,Y. start/end are positions of bps just BEFORE first and AFTER last Indel gap in any genome, so they refer to the same two bps in all genomes. Always start < end. If - phase, start is bp just AFTER, end is bp just BEFORE, opposite of +. Length of the Indel region in each genome is end-start-1.

Table 6. Columns in *.snps.tsv files; X,Y=chosen genome letters

Column	Description
ID	Unique ID tying row back to originating input file row. LCR input files: LCRnumber. IndelGroup and Markers files: refID_refPos1_refPos2.
phases	Phase of each genome incl. ref. genome, relative to ref. genome, string of +/- chars, + : same direction, - : opposite direction.",
idx	Starts at 1 and counts each SNP within an ID. For given ID (input row), number of SNPs in that region is max idx value.
Xid,Yid	Sequence ID of the SNP in genomes X,Y.
Xpos,Ypos	SNP position in genomes X,Y.
Xval,Yval	SNP value in genomes X,Y.

Table 7. Columns in *.withseqs.tsv files; X,Y=chosen genome letters

Column	Description
ID	Unique ID tying row back to originating input file row. LCR input files: LCRnumber. IndelGroup and Markers files: refID_refPos1_refPos2.

Column	Description
phases	Phase of each genome incl. ref. genome, relative to ref. genome, string of +/- chars, + : same direction, - : opposite direction.",
Xid	Genome X sequence ID
Xpos1	Genome X position of upstream end of sequence to align on "+" strand
Xpos2	Genome X position of downstream end of sequence to align on "+" strand, Xpos1 < Xpos2 always
Xseq	Genome X DNA sequence between the two positions