Overview

phyloGenerator 2 is the “second version” of the program phyloGenerator. The intention of phyloGenerator 1 was to make it easier for ecologists to generate their own phylogenies for use in their own work. If you are such a person, I would still advise you to use phyloGenerator 1, which is is still supported and maintained (the latest version was released in September 2016). Many users wanted to go beyond what phyloGenerator 1 was designed for. For example, they wanted to build very large phylogenies—phylogenies of thousands of species, or they wanted to build phylogenies for groups that have never been studied before. While that is possible using phyloGenerator 1, doing so requires skills that phylogeneticists have. There is a reason the field of phylogenetics exists, and the reason is that such tasks are hard.

phyloGenerator 2 is a complete re-implementation of phyloGenerator 1, designed for use by phylogeneticists who want to build large trees. It automates the process of sequence download and selection, making use of multi-threading to do it much faster than phyloGenerator 1. It is very easy to go wrong with phyloGenerator 2: the code is much leaner, and so faster, and so it doesn’t perform many of the checks that pG1 performed. For example, it will not complain if you don’t give it a constraint tree. You cannot check to see if your loci are not concordant. If you’re attempting to build a phylogeny of thousands of species, you are going to have to do some leg-work to check that your phylogeny is a reasonable hypothesis, and that your data are reasonable. There is only so much I can reasonably attempt to automate, and pG2 reflects my best attempt. Good luck!

Remember you can always ask questions on the phyloGenerator mailing list (`https://groups.google.com/forum/#!forum/phylogenerator-users`) and the website has lots of tips and tricks to get you going (`http://willpearse.github.io/phyloGenerator2/guide.html`). I do like talking to people, and I do try to be nice to them while I do, so reach out if you’re having a problem.

Giving birth: installation

I assume, below, that you are on a UNIX-like system (*e.g.*, MacOS or Linux). If you’re on Windows 10, activate “developer mode” and you will be able to get a Linux Terminal within Windows. A great tutorial for doing this is avaiable online (`http://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/`) thank you Anna Alessi for suggesting this! Once you have that, follow the instructions below. I cannot reasonably be expected to support every single operating system: I consider Windows 10, any version of MacOS X, and Linux to be a lot! An alternative in Windows, which I give in the appendix, is to use `cygwin`, which is a sort of UNIX-like thing for Windows. Adriana de Palma produced this—thank you Adriana! Please contact me if you have any problems with the following installation steps. I assume a basic level of competency (*e.g.*, I don’t explain how to move between directories). If you’re not comfortable doing things like this, you’re not going to have a good time with pG2 and so I suggest you try pG1 to begin with.

1. Open a terminal window.
2. **Install Ruby and Git**.
   On Linux/Windows, run `sudo apt install ruby`, but in your password, and you’re done. On MacOS, you should already have Ruby installed. To install `git`, type `git` into Terminal: you will be asked if you want to install Developer Tools—you do, so follow the on-screen instructions.
3. **Install BioRuby**.
   Type `sudo gem install bio`, enter your password, and you’re done.
4. **Install MAFFT**. On Windows or Linux, type `sudo apt install mafft`, and enter your password. On a Mac, go to the MAFFT website (`http://mafft.cbrc.jp/alignment/software/`), download the appropriate version for your computer, and install.
5. **Install RAxML, ExaML, or ExaBayes**.

To build a phylogeny, you need some sort of phylogenetic search program, and pG2 supports two out of the box: RAxML, ExaML, and ExaBayes. I'll assume you want to use RAxML: enter `git clone https://github.com/stamatak/standard-RAxML.git`, then go into the directory that creates and `make -f Makefile.AVX2.PTHREADS.gcc` or whatever implementation makes sense for you (note there's a `make -f Makefile.AVX2.PTHREADS.mac` for Macs). You now need to put the resulting executable somewhere on your path: for me, `sudo mv name.of.executable /usr/local/bin/raxml` (note the executable must be called `raxml`) did the trick. The same steps need to be followed for the other two programs, and pG2 needs `parse-examl` and `examl` to be on your path for ExaML, and `yggdrasil` for ExaBayes.

6. (Optional: **Install ape**).
   If you're using ExaML, you need R and ape installed to get a starting tree for your search. Open up R (if you don't have R installed, you'll be much happier in general once you do, honest) and run `install.packages("ape")`. R (specifically Rscript) needs to be runable from the Terminal; if it's not, check you've installed R properly.
7. **Install phyloGenerator2**.
   Run `git clone https://github.com/willpearse/phyloGenerator2.git`.

Voila!

First steps: plants

pG2 expects you to give it a configuration file, and once you've given it that it'll do the rest for you. Take a look inside the "simple plants" folder in "demo". There you'll see `params.yml`—open it up. This is a YAML file (YAML 'ain't a Markup Language: it's a stupid acronym, I know...), and the format is very simple: you can comment out lines with `#` and a colon is used to indicate a parameter value. Go through the "basics" section at the top of the document, adding your email address, the location on your computer where all the files are, and an existing folder where you want the output to be put. Every file location in pG2 should be a full, absolute path, so on a PC it should start with something like `C:` and on everything else `/`).

(Read only if you open a parameter or Ruby file and it looks strange). I'm a Linux user in the USA, which means if you're in another country or on another operating system (Windows) when you open my files they may look a bit odd on your computer. `pG2` won't be affected by this, but the text editor on your computer may be when opening demos. If you have this problem, try opening the file in a text editor like Notepad++.

The rest of the file is reasonably self-explanatory. Your main task is to set up the "gene block": the part of the file where you list what genes (loci, technically) you want pG2 to download, and parameters that define the search procedure. Let's skim over the details for now, and focus on one thing: the *reference file*. This is a file containing exemplar sequences that pG2 will use to guide its search (in pG1 this was called the `referenceDownload` option). If you don't give a file here, then pG2 will just download the first thing it sees. Don't worry about the options here, but make sure you get the general principle that there's a section for every gene, where the name of the section is the thing pG2 will be searching for (in this case, *rbcL* and *matK*).

Outside of the gene block, you can specify options to pG2. Make sure you comment out whichever construction method you don't want to use (you can only use one per run). We'll go through the other options in detail in the next section.

You're now ready to run the program! Open up Terminal, and run something like this: `ruby pG_2.rb /full/path/to/params.yml`. Obviously, you should replace `/full/path/to/params.yml` with the full path to your params file. Press enter, and wait... *If you get an error message, don't panic*. In the nicest way possible, you've probably not followed all of my instructions above. Before contacting me, read the error message and see if you can figure out what it means. For example, an error relating to a badly formatted configuration file means your configuration file has an error in it. A "file not found" either means a file isn't on your hard-drive, or you've given pG2 the wrong location of it (did you use a full, absolute path?). If you can't find the solution, by all means copy-paste what you find in your Terminal window into an email to the mailing list, along with the configuration file you used.

After a little while, you should have two folders in the working directory you specified. One of them contains all of the sequences used in the format `genus_species_gene.fasta`. I picked this formatting for a reason: you can use something like `ls genus_species*` to get all the sequences for a species, and `cat *gene.fasta > gene.fasta`to make a combined set of sequences for a particular locus. The other folder contains all of the gene alignments, and the raw output from your phylogeny building step. Take a look at all of these outputs, and see if you're satisfied with the resulting tree.

Walking: zooplankton

OK, so we can build a plant phylogeny. What about something a bit less common? What about... zooplankton? In the zooplankton folder there's a demo from a real species list that I was sent by someone—this isn't a list that I've made up to make the program look good, but rather something someone contacted me to try and get me to be a co-author by building them a tree. Instead, I added it to the program and they've got it for free! Everyone's happy :D It's a species list of 268 species, and you're going to build a six-locus phylogeny of those species. It's not going to be perfect: you will have to examine it and see how pG2 does.

To get the most out of the program, you're going to have to change the parameters somewhat. The most important section is the "gene" section. Below, I go through each option in turn. For a minute, don't worry if you see the word "hawkeye"—we'll get to that in a minute.

- **ref_file**—the location of your reference sequence file (FASTA format), as described above)
- **ref_min**—the minimum length a downloaded sequence must be
- **ref_max**—the maximum length of an alignment of your sequence when aligned with `ref_file`. This will also be used for the hawkeye method (see below).
- **max_dwn**—maximum number of sequences to try for each species
- **fussy**—(optional) if present and set to false, don't use GenBank gene and organism annotations, and don't attempt to trim a sequence down using GenBank annotations.
- **aliases**—an array of alternative names for your gene. For example, if your block was called COI, and you specified aliases = [cytochrome oxidase one, cox1], pG2 would search for COI, cytochrome oxidase one, and cox1. You must use the square brackets ([]) otherwise bad things will happen.
- **max_gaps**—(hawkeye only) maximum number of gaps in sequence when aligned.
- **gap_length**—(hawkeye only) length that defines a gap in `gap_length`.

Most of these are self-explanatory, and refer to the options given to *referenceDownload* when it's checking sequences. The more confusing options refer to gaps. Gaps are stretches of sequences that, when aligned, contain only gaps (`-`). A few of them is fine, but long stretches of them indicates that a particular sequence isn't of good quality, or isn't the locus it claims to be. The purpose of *hawkeye* is to spot long gaps, and by setting a tolerance on how many and how long you'll permit those gaps to be for a particular sequences, you can fine-tune how pG2 works. For example, a very well-behaved region that evolves slowly like, say, *rbcL* or *COI*, shouldn't have very many gaps. However, something that evolves a lot faster, like *matK*, will. There's no magic shortcut to knowing what these options should be—you'll have to experiment.

pG2 is fast because it's simple. It relies on very basic operations like aligning very small subsets of your data, and regular expressions to find gaps, because those are operations that have reasonably high power to get rid of obviously dreadful sequences, pretty low error rates at pulling out things that are definitely correct, and are so fast that they can be done lots and lots of times with very little memory. I have tried lots and lots of more complicated methods, and honest these are the ones that work, and they don't require thousands of hours of super-computing time. You will have to check your alignments once they are built, and chuck out the handful of sequences that "slipped through". This is not a mistake: it is by design. If I had automated this entire process in a foolproof manner, I would have built a unified

Tree of Life by now. If someone else had done so, they would as well. If you have other ideas, please get in touch!!! :D

Now, what about the other options hiding at the bottom of the parameter file?

- **hawkeye**.
  If present and set to true, runs the new hawkeye sequence check. The parameters are described in the last section, and the new method is described at the top of this page. If you use this option, you'll get a new folder (hawkeye) in your output, with a load of species that are re-named to mark them as 'bad'. Anything still in the seqs folder has passed the hawkeye check.
- **cache**.
  Skip DNA download for all species that have at least one DNA sequence (as outputted by pG2) in this folder. Note that a single sequence will be sufficient to stop further searches. There's no point in searching for the same species over and over again if you didn't find anything the first time and the settings haven't changed; just remove those species from your species list the second time.
- **phy_method**.
  Either `raxml`, `examl`, or `exabayes`, depending on which program you want to use to build you phylogeny. You don't have to supply this option; giving nothing will just download sequences. If using ExaBayes, bear in mind that pG2 does no summarising of the posterior, unlike pG1. The reason for this is simple: despite my warnings, people who had no idea how to conduct a Bayesian analysis were using pG, and so many were not checking for mixing, convergence, that they had sufficient samples, etc. I then received a number of very angry emails from people who hadn't followed instructions (and had often ignored pG1's warnings) that their results were "obviously false". I don't enjoy being shouted at, so I dropped support for this.
- **constraint**.
  Location of (Newick) constraint tree to be used in your build. Note that only RAxML supports a constraint tree at this time, and nothing supports any kind of dating (yet; I plan to add it). Your constraint tree must contain all of the species you are attempting to download data for, but if you can't find some data for some species pG2 will just drop those species for you.
- **partition**.
  If true, a separate rate matrix will be fit to each locus, along with gamma parameters et al. I can't imagine you would want to set this to false.
- **phy_options**.
  Each phylogenetic construction program has special options that can be passed to it. I've not written a parser for each program, as I did in pG1, but I have written some examples of common things you might want to do that essentially replicates the work of pG1. Experience tells me most users want to do something I hadn't thought of; you wanted the power, now you've got it!

I've been through the *hawkeye* option, but you'll almost certainly want to consider using the `constraint` options as well as the `cache` options. Constraint trees are useful because they let you ensure that your search doesn't consider tree topologies that you are certain are not true. However, if you're relying on this option too heavily, consider whether you need to build a tree (what's the tree you're using as a reference? will it do for your purposes?) and whether the data you've grabbed are of sufficiently high quality for your to rely on them. The cache option is also a god-send: it lets you keep a load of sequences that you're already comfortable with and pG2 will use these sequences in the next run and then only go to GenBank for sequences if it's missing information from this option. Bear in mind that it assumes that you're done with a species if it finds *anything* for it here. It's easy to just copy-paste sequences from one cache folder to another, so this is a great way to save work from previous runs (and search for one locus at a time and then put everything together at the end, if you're so-inclined).

Run the zooplankton demo. What do the branches look like on the resulting phylogeny? Are close relatives coming out together? Are any species on particularly long branches? Can you find alignments that are too gappy, and so should have their download/Hawkeye options changed?

Advice for joggers

Building phylogenies is hard, and takes time and a little bit of patience. pG2 will make things easier for you, but if you're trying to build a phylogeny of over a thousand species with several loci please don't be frustrated if the process ends up taking longer than an afternoon. Below are a few tips to get you started on your own projects.

- Start small. You don't have to build a phylogeny of a thousand species the first time: focus on a smaller number (*e.g.*, 50) and slowly work up from there.
- Use the cache. It takes a long time to download sequences, so once you've got them, keep them.
- Filter out sequences. It's easy to add/remove sequences to a cache folder, and use that to your advantage.
- Build up the complexity. Don't use *hawkeye* for your first run. Once you've got a load of sequences, and an alignment, see what happens when you run *hawkeye* on the cached sequences. Experiment with different options for *hawkeye*—sometimes it works well, and sometimes it doesn't.
- Don't be afraid to fly solo. Some users find pG2 is a great way to get sequences, and then go off on their own from there. That's fine: if you just want to use it for download, I'm not offended—it's what I do! :D Remember you can hit control-c during the phylogenetic building step to stop the program, and all your sequences will be sat in your working directory waiting for you (along with a few files pG2 makes while running, like program logs and alignments).
- You will find yourself frequently making new YAML files for runs, each of which will want to output to different directories. So while my examples use files called `params.yml`, you'll probably want to make files called `params_1.yml`, `params_2.yml`, etc. Of course, you'll also want to use version control, but that's another story...

Appendix: `cygwin` on Windows

- Open Cygwin Terminal
- Check ruby is installed (`ruby -v`)
- Check gem is installed (`gem environment`)
- Install bioruby (`gem install bio`)
- Download and extract mafft (follow instructions here: `http://mafft.cbrc.jp/alignment/software/windows_cygwin.html`)
- Clone RAxML from git (remember to save it the directory you want, otherwise it will save to Cgywin dir)
- Place the executable in: `C:\cygwin64\usr\local\bin`
- Clone phylogenerator2 from git and follow instructions above!

Thank you Adriana de Palma who wrote the above!!!