Plans for the Phylogenetic Comparative Methods Benchmark Database (PhyCoMB)

September 8, 2017

CONTENTS

1 Introduction

This document is intended to be a high-level description of the functionality and structure of what PhyCoMB will eventually look like. Ultra-briefly: The goal is to allow users to compare the performance of different phylogenetic methods. There will be a web interface that allows for browsing those performance results, and for contributing new results. Behind that will be a database of the testing datasets, methods, and results.

2 Original Proposal

Here is what was originally proposed to NSF/BSF. It is still pretty much what I have in mind, but the subsequent sections provide much more detail.

PhyCoMB will challenge new methods in a standardized manner, revealing their strengths and weaknesses early in their lifecycle. It will focus on testing if discrete traits affect rates of speciation and extinction, but the work will also support questions of trait evolution and lineage diversification separately.

2.1 Community resource for benchmark tests of model performance

A paper introducing a new phylogenetic comparative method typically includes essential simulations that examine its power and bias and reveal the parameter space in which it will be most beneficial. These simulations typically follow the assumptions of the underlying model. Poor behavior, however, may arise when those assumptions are not met. Because all empirical datasets have been shaped by processes outside the assumptions of any model, it seems self-evident that methods should routinely be tested in many situations beyond their specific focus. Why is this not standard practice for comparative methods developers? It is time-consuming or impossible for a single developer to craft diverse testing datasets that encompass the biological phenomena likely to 'break' his/her new method. Furthermore, there is not a culture—during development, peer review, and empirical application—of valuing and hence requiring robustness testing in this field. Thus, phylogenetic comparative methods papers typically discuss possible artifacts but fall short of providing concrete guidance about whether a method can be reliably applied to data at hand.

We aim to break these barriers to the routine, rigorous testing of new phylogenetic comparative methods by making robustness testing more straightforward. We will develop a suite of tests that can easily be deployed to assess the performance of a new method and compare its behavior against other methods designed for the same questions. The product will be called **PhyCoMB: Phylogenetic Comparative Methods Benchmarking** (Fig. 1). Benchmark tests are standard practice in many fields, from computer hardware to bioinformatics. For example, BAliBASE (the Benchmark Alignment dataBASE; Thompson et al. 1999, 2005) has been widely adopted for assessing the performance of multiple sequence alignment algorithms. Further examples of standardized benchmark tests are Assemblethon and

BUSCO for genome assembly (Bradnam et al. 2013; Simão et al. 2015) and T. Warnow's resources for phylogeny estimation (*http://www.cs.utexas.edu/~phylo/datasets*). PhyCoMB's structure will mirror other benchmark suites:

◦ *Tasks* are specific questions within the domain of phylogenetic comparative methods. They include tests of whether evolving traits affect rates of speciation and extinction (our focal task here), clade-specific diversification rate shifts, irreversible evolution, and discrete trait correlations.

◦ *Methods* are procedures designed to accomplish a task. They consist of a model or other technique (e.g, BiSSE, sister clades), plus a statistical inference framework (e.g., AIC, model averaging, sign test).

◦ *Elements* are collections of trees and optionally traits, all with the same properties. They may arise from empirical or simulated data. A method is applied to each tree/trait item within an element.

◦ *Reference sets* are chosen to present particular challenges to a method, within the context of a task. Each is a group of elements. For example, the 'power' reference set includes elements with small and large trees. The 'pseudoreplication' reference set contains trees in which traits change rarely and are accidentally associated with diversification shifts. The 'diversification heterogeneity' reference set contains trees with complex shifts in speciation, on which neutral traits are evolved.

◦ *Curated benchmarks* are the heart of PhyCoMB. The underlying database will eventually contain dozens of reference sets with many thousands of elements, but we will carefully select a subset to be the benchmark test for a task. For example, the benchmark for trait-dependent diversification might consist of 16 elements forming a progression of increasingly challenging power tests, 24 elements with different forms of pseudoreplication, and 36 elements with different forms of diversification heterogeneity and neutral trait evolution. Manual curation will maximize the diversity of challenges faced by a method while keeping the amount of testing manageable. Clearly-defined benchmarks will allow the performance of different methods to be easily compared by those developing new methods and those applying them to empirical systems.

◦ *Annotations* will make the results obtained from PhyCoMB easier to explore and interpret. Each element will be labelled with, e.g., the number of tips and sampling completeness, the type of diversification process for simulated trees, construction methods for empirical trees, and the model of evolution for simulated traits. Contributors will note the origin of the element's data (literature citation or generating script).

◦ *Reports* will allow developers to see how a new method performs relative to others, and they will allow users to understand the strengths and weaknesses of a variety of methods for a given task.

For an empiricist, PhyCoMB will provide not only structured information about methods, but also the means for directed testing. This includes scripts for creating test data that reflect properties of real data, and for running methods on them. It can thus enhance the impact of empirical work by aiding demonstrations of the robustness of a method for the data and question at hand. Such improved connections between methods development and empirical use will bring greater stability and transparency to the field (Cooper et al. 2016).

The first life-stage of PhyCoMB will be initiated by this proposal. We will generate a collection of elements and benchmarks (soliciting ideas during the workshop; see Broader Impacts), and we will build a web-based interface for developers and end users. This first stage will demonstrate substantial benefits to the phylogenetics community: straightforward comparisons of old and new approaches, reproducibility of results, early warnings of methodological weaknesses, and highlights of when methods are particularly powerful and robust. The second stage that we envision for PhyCoMB is gradual community adoption beyond the timeframe of the proposed work. PhyCoMB will evolve through contributions of new elements, as future work uncovers complex scenarios that challenge the assumptions of our methods.

2.2 References

Beaulieu, J.M., B.C. O'Meara, 2016. Detecting hidden diversification shifts in models of trait-dependent speciation and extinction. Systematic Biology 65:583–601.

Bradnam, K.R., J.N. Fass, A. Alexandrov, P. Baranay, M. Bechner, I. Birol, S. Boisvert, J.A. Chapman, G. Chapuis, R. Chikhi, et al., 2013. Assemblathon 2: Evaluating de novo methods of genome assembly in three vertebrate species. GigaScience 2:1.

Cooper, N., G.H. Thomas, R.G. FitzJohn, 2016. Shedding light on the dark side of phylogenetic comparative methods.

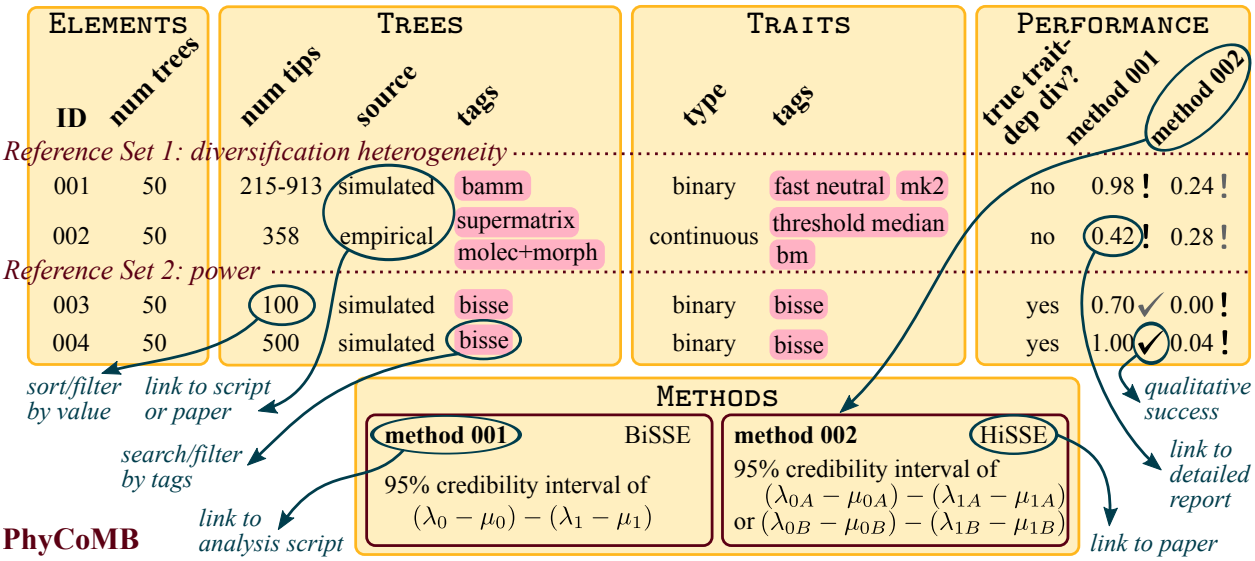

Figure 1: Vision for PhyCoMB. The Phylogenetic Comparative Methods Benchmark database will provide test datasets and reports of method performance. Example annotations are shown for four elements (comprising trees and traits), and interactive behavior of the web interface is noted. Compared are two methods, which test for trait-dependent diversification by assessing the difference in estimated net diversification rates. Performance is reported as the proportion of trees on which significant trait-dependent diversification is inferred. Reference set 1 reveals the BiSSE-based method is prone to incorrectly associate two kinds of neutral traits with diversification rate, while the HiSSE-based method is less so. Reference set 2 reveals that HiSSE has much lower power than BiSSE in this test. (Note that HiSSE is much more effective under model averaging; Beaulieu & O'Meara 2016.)

Methods in Ecology and Evolution 7:693–699.

Simão, F.A., R.M. Waterhouse, P. Ioannidis, E.V. Kriventseva, E.M. Zdobnov, 2015. BUSCO: Assessing genome assembly and annotation completeness with single-copy orthologs. Bioinformatics 31:3210–3212.

Thompson, J.D., P. Koehl, R. Ripp, O. Poch, 2005. BAliBASE 3.0: Latest developments of the multiple sequence alignment benchmark. Proteins: Structure, Function, and Bioinformatics 61:127–136.

Thompson, J.D., F. Plewniak, O. Poch, 1999. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. Bioinformatics 15:87–88.

2.3 Later thoughts

This subsection wasn't part of the proposal. But it explains some ways in which the current plan differs from what was proposed.

Task needs to be defined a bit more carefully. For example, even within the realm of state-dependent diversification, one approach/question is hypothesis testing and another approach/question is parameter estimation. Results for those questions would be reported differently. And even a given method on a given element could perform better on one question than the other. Below, *Task* is assumed to include this refinement. But we could instead have a separate *Question* component that intersects with *Task* to specify the problem.

If we are going to include methods that co-infer the phylogeny along with answering the comparative methods question, we'd need to allow for input data to be a sequence alignment rather than a tree. That is a potentially promising direction for methods development, but it's beyond the immediate scope of PhyCoMB.

3 USERS

Different kinds of users will interact with PhyCoMB in different ways. They will have different goals and permissions. One person could be a different type of user at different times, e.g., I could do some work as an Administrator, then add data as a Contributor, than look through results as a Viewer.

3.1 Viewer

Can browse PhyCoMB contents through the web interface. Cannot make changes that affect anyone else. No login required.

3.2 Contributor

All the functionality of a Viewer. Additionally, Contributors can see and use the parts of the web interface for uploading *Element*s, *Method*s, and *Performance*. Login required.

Should the newly uploaded material be immediately integrated into the database and visible to all users, or should it not be published until an Administrator approves the submission?

Contributors can't delete information, but perhaps they could flag *Element*s, etc. that are no longer useful, so an Administrator could delete them.

3.3 Administrator

Can approve the uploads and suggested deletions made by Contributors. Can also make changes to the structure (e.g., which *Element*s are in a *BenchmarkSet*). Perhaps the interface will be different than what a Viewer or Contributor sees. Login required.

4 WORKFLOWS

Here are some examples of how different kinds of users might commonly interact with PhyCoMB.

4.1 Viewer workflows

4.1.1 Explore performance of various methods for a particular task

The user's goal is to learn about how various methods perform for questions they're interested in. This means seeing which *Method*s are available for a given *Task*, and their *Performance*, summarized in a *Report*. This is the most important user interface component to design well.

1. Select the desired *Task*. See **??** for what that might look like.
2. Select the desired *Method*s. See **??** for what that might look like.
3. Select the desired *Element*s. See **??** for what that might look like.
4. Browse the *Report*. See **??** for what that might look like and the types of interactivity needed.
5. Download the *Report*. See Sect. 7.3.

4.1.2 Explore performance of a particular method for various tasks

The user's goal is to learn about how a method they're interested in performs for various questions. This means seeing which *Task*s are addressed by a given *Method*, and its *Performance*, summarized in a *Report*.

1. Select the desired *Method*.
2. Select the desired *Task*, or default to all relevant *Task*s.
3. Select the desired *Element*s. See **??** for what that might look like.

4. Browse the *Report*. See **??** for what that might look like and the types of interactivity needed.
5. Download the *Report*. See Sect. 7.3.

4.2 Contributor workflows

4.2.1 Contribute a new method

The user's goal is to add a *Method* to the ones available, thinking that it will be useful for other people.

1. Browse existing *Method*s to determine if it's already available.
2. Provide a script that completely runs the *Method* when provided with an *Element*, returning the value for *Performance* of a *Task*.
3. Fill in a form with basic information. This info helps to categorize the *Method* in the database, and it will be passed along to future users who see the *Method*.
   - Specify what *Task* it is for. Maybe also what *ReferenceSet* it is most interesting for, e.g., if method was designed to fix a particular weakness.
   - Explain why the new *Method* is worthwhile—what was the purpose in creating it?
   - Provide info that becomes values in columns: what category of model/technique, what kind of statistical inference, etc. (see **??** for the required fields).
4. Upload *Performance* results for at least the *BenchmarkSet* of *Element*s.
5. See a *Report* to confirm that things look correct.
6. Request that an Administrator accept the new *Method*?

4.2.2 Contribute a new element

The user's goal is to add an *Element* to the ones available, thinking that it will be useful for other people.

1. Browse existing *Element*s to determine if it or something similar is already available.
   (a) If the new *Element* is better than an existing one, include a deprecation request for the Administrator?
2. Provide data files for the new *Element*.
   (a) Generating script, if applicable. Link to empirical data source. etc. Needs to be completely documented and reproducible.
   (b) The actual tree and trait files. Or if using *Tree* and/or *Trait* from another *Element*, link to that.
3. Fill in a form with basic info: This info helps to categorize the *Element* in the database, and it will be passed along to future users who see the *Element*.
   - Specify what *Task* it is for, and what *ReferenceSet* it will belong to.
   - Explain why the new *Element* is worthwhile—what was the purpose in creating it?
   - Provide info that becomes values in columns: simulated/empirical, sim regime, etc. (see **??** for the required fields).
   - Can some info be obtained automatically from the uploaded files? e.g., tree size
4. Upload *Performance* results for all(?) *Method*s that apply to that *ReferenceSet*.
5. See a *Report* to confirm that things look correct.
6. Request that an Administrator accept the new *Element*.

4.2.3 Contribute a new performance result

This could be part of the workflow in contributing a new *Method* or *Element*. But it could also be done separately. In that case, find combinations of *Method* + *Element* that haven't yet be run, and fill in the values.

4.3 Administrator workflows

4.3.1 Alter the groupings of elements

Alter the *Element*s in a *ReferenceSet*. Alter the *Element*s and/or *ReferenceSet*s in a *BenchmarkSet*.

4.3.2 *Approve an addition from a Contributor*

Check the new *Method* or *Element* and publish it (make it available to all users) if it looks good. Maybe a general form for other requests, e.g., to deprecate an *Element*?

4.3.3 *Deprecate a method or element*

Don't entirely delete from the database, but hide from most operations?

4.3.4 *Manage users*

Field requests from users who wish to become Contributors? Or just let that be automatic, and then revoke Contributor status if someone causes trouble?

5 VIEWS [TODO]

6 DATABASE TABLES

I'm assuming the information would be stored in a relational database. Here's what could be in the tables.

I think it's realistic to settle on the set of tables and their relationships early on. But some of the content in the tables (columns, allowable values for some columns) will probably be adjusted as we go along.

6.1 *Task*

Unique ID: arbitrary number. Each *Task* will be identified to a User by a phrase. There won't be many *Task*s, and perhaps only one for awhile.

Do we also need Questions or other attributes within each *Task*? For example, say my Task is state-dependent diversification. Discrete traits versus continuous traits. Hypothesis testing Question versus parameter estimation Question.

Questions and Tasks could be separate tables. Each *Task* could include one or more Question.

6.2 *ReferenceSet*

Unique ID: simple human-readable numbering scheme. There will also be a phrase that identifies each one to a User.

Task (s) it's relevant to.

Number of *Element*s it contains? Can be determined from *Element*s table.

6.3 *BenchmarkSet*

some *Element*s, which bring along their *ReferenceSet* membership

6.4 Elements

Each *Element* consists of one *Tree* (Sect. 6.5), optionally one *Trait* (Sect. 6.6), and some other information.

Unique ID Arbitrary, e.g., E-47295. Auto-generated when created.

Tree Link to corresponding *Tree*.

Trait Link to corresponding *Trait*, if any.

Refset Link to one or more *ReferenceSet*s for which this *Element* is relevant.

Number of items Positive integer. It's determined by the combination of *Tree* and *Trait*, but the Contributor will provide this info.

Contribution info (could be same or different from info for *Tree* and/or *Trait*)

Contributor a registered user, see Sect. 3.2

Date auto-populated when *Element* is created

Comment a few sentences provided by the Contributor

It will be common for a user to download one or more *Element*s (Sect. 7.1).

6.5 Trees

Each *Tree* object is actually a set of trees, all with the same properties. Here are those properties:

Unique ID Arbitrary, e.g., T-83247. Auto-generated when created.

The trees Each individual tree is itself stored as a Newick string. Those strings could reside directly within the database; they can be quite long, though, which might be troublesome. Or the database entry could be a link to text file(s) containing the trees; this might be better because such files will frequently be downloaded by users (Sect. 7.1).

Generating script The code used to simulate or otherwise create the trees. A file to download. Not all *Tree*s will have one.

Number of trees Positive integer.

Contribution info :

Contributor a registered user, see Sect. 3.2

Date auto-populated when *Tree* is created

Comment a few sentences provided by the Contributor

Elements The *Element* (s) to which the *Tree* belongs.

Columns of tree info Will be figured out as we go along, but likely ones are:

Number of tips single number or numeric range

Source 'simulated' or 'empirical'

Tags Various descriptive words. The idea with tags is that they are not necessarily alternatives (like 'simulated' versus 'empirical'), and there could be any number per *Tree*. With use, we might realize that some tags can be converted to columns, and maybe vice versa. Tags probably involves two extra database tables: (1) columns are TagID and TagName, one row per tag; (2) columns are TreeID and TagID, one row per tag per tree.

6.6 Traits

Each *Trait* object consists of at least one trait value per species. There could be multiple such sets in one *Trait* object. In that case, all the trait sets have the same properties. Here are those properties:

Unique ID Arbitrary, e.g., A-57387. Auto-generated when created.

The traits Each set of traits is simply a list of numbers, labeled by tip/species name. As for *Tree*s (Sect. 6.5), this info could reside directly within the database or in a linked text file (e.g., CSV), which will frequently be downloaded by users (Sect. 7.1).

Generating script The code used to simulate or otherwise create the traits. A file to download. Not all *Traits* will have one. Might be the same as the generating script for the corresponding *Tree*.

Number of trait sets Positive integer.

Contribution info (might or might not be identical to the corresponding *Tree* (s))

Contributor a registered user, see Sect. 3.2

Date auto-populated when *Trait* is created

Comment a few sentences provided by the Contributor

Trees The *Tree* (s) to which the *Trait* corresponds.

Columns of trait info Will be figured out as we go along, but likely ones are:

Numerical type 'discrete' or 'continuous'

Source 'simulated' or 'empirical'

Tags Various descriptive words. (See tag notes in Sect. 6.5. Different tags for *Tree*s and *Traits*, though.)

6.7 *Method*

Unique ID: generate as each *Method* is created. Users will see that identifier, so should be human-readable. Could be arbitrary numbers or letters, either generated sequentially or randomly (to reduce user bias). Or could reflect the content of the *Method*, e.g., first letter of model/technique and statistics.

Task (s) it's relevant to. Or only *ReferenceSet* (s) it's relevant to, and then that is mapped to *Task* (s)?

Model (bisse, etc) or Technique (sister clades). Statistical approach (bayes factors, aic, model averaging, sign test, etc). (These should be categories. Maybe their own Table? With links to papers/citations?)

Script. Should take *Element* as input, return a simple answer to *Task*.

How to run a single analysis and return results in multiple forms, e.g., hypothesis testing question and parameter estimation question? Call those separate *Method*s? Link those *Method*s for clarity and code reuse?

- Name of the Contributor.
- Comment from the Contributor.
- Date contributed.

6.8 *Performance*

Unique ID: arbitrary number.

Each combination of *Method* and *Task* (or at least Question) has a unique *Performance* result. Hopefully this can be just a single number, e.g., proportion of trees on which the conclusion was correct.

But we might need more, e.g., accuracy of estimates for various parameters. This could depend on both the *Method* and the Question.

7 DOWNLOADS

After users view and filter the results, they may want to download information for use offline. Each download should be a single zipped file, which contains only plain text files with a logical directory/folder structure.

7.1 Obtaining testing files

When an *Element* is downloaded, the user should receive:

- Information about it (Sect. 6.4), written in an auto-generated text file.
- *Tree* files and/or generating script and information (Sect. 6.5).
- If applicable, *Trait* files and/or generating script and information (Sect. 6.6).

These files should all have sensible names.

When multiple *Element*s are downloaded together, each should be in a separate directory. If they have tree or trait files in common, could have an option to use symlinks instead of duplicating the content. Also, a spreadsheet (CSV file, one row per *Element*) should be included so it's easy to see which *Element*s have which attributes (columns in *Tree*, membership in *ReferenceSet*, etc.).

If an entire *ReferenceSet* is requested for download, each *Element* within it should be in a separate directory.

If an entire *BenchmarkSet* is requested for download, each *ReferenceSet* within it should be in a separate directory.

Need to decide on the file format for *Tree* and *Trait*. Some options:

- One Nexus file per *Element*, containing all the trees and all the traits. Uncluttered, but more annoying to parse.
- One file per *Tree* (each line a Newick string) and one file per *Trait* (CSV with one column per trait).
- One file per tree (many per *Tree*) and one file per trait (many per *Trait*), with filenames that show which belong together (e.g., `t001.tre` and `s001.csv`).

7.2 Obtaining methods scripts

When a *Method* is downloaded, the user should receive:

- Information about it, written in an auto-generated text file.
- The script to run it.

When multiple *Method*s are downloaded together, each should be in a separate directory.

7.3 Obtaining performance reports

The user should be able to download a CSV file that looks basically like the results table *Report* (**??** or **??**). Either the full report could be requested, or only to include those rows (*Element*s) and columns that are visible after interacting with the report view.

Are additional columns needed, e.g., directory names of *Element*s and *Method*s if they are downloaded?