Plans for the Phylogenetic Comparative Methods Benchmark Database (PhyCoMB)

July 8, 2019

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 (described roughly in Sects. 4 and 5). Behind that will be a database of the testing datasets, methods, and results (described in some detail in Sect. 6).

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

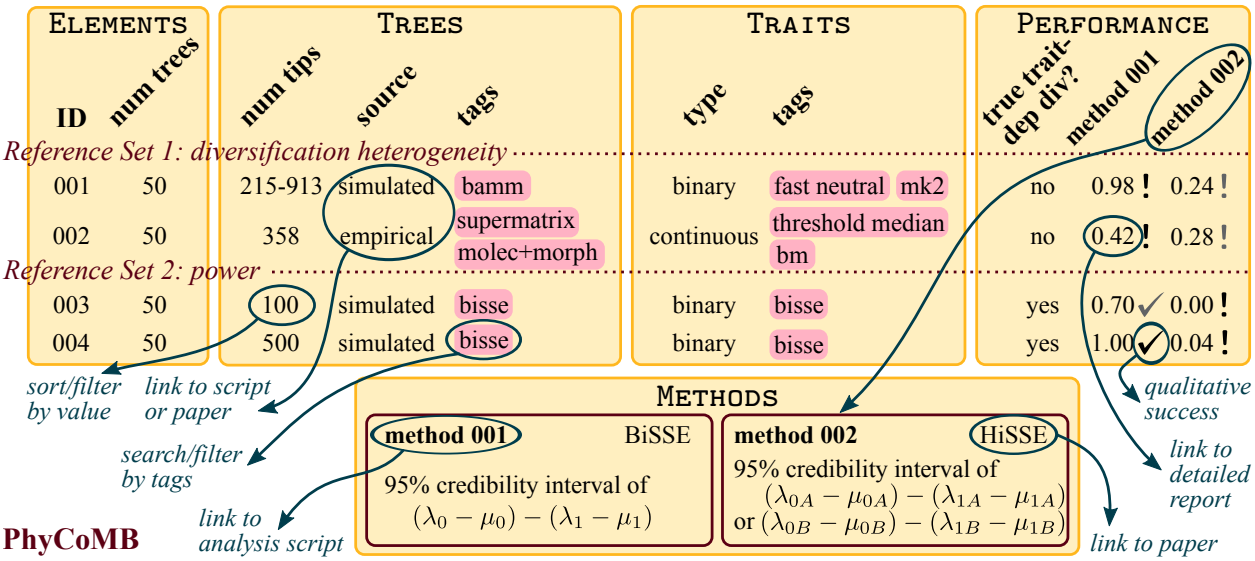

Figure 1: Vision for PhyCoMB. The Phylogenetic Comparative Methods Benchmark database will provide test datasets and reports of method performance. Example annotations are shown for four elements (comprising trees and traits), and interactive behavior of the web interface is noted. Compared are two methods, 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.)

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. 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 *TestElement*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 *TestElement*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, like which *TestElement*s are in a *BenchmarkSet*. Perhaps the interface will be different than what a Viewer or Contributor sees (e.g., Django provides a nice Admin interface semi-automatically). 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*.
2. Select the desired *Method*s.
3. Select the desired *TestElement*s.
4. Browse the *Report*. See Sect. 5.2 for 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 *TestElement*s.
4. Browse the *Report*. See Sect. 5.3 for 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 *TestElement*, returning the *Result* of a *Task*.
3. Fill in a form with basic information (described in Sect. 6.7). This info helps to categorize the *Method* in the database, and it will be passed along to future users who see the *Method*.
4. Upload *Result*s for at least the *BenchmarkSet* of *TestElement*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 *TestElement* to the ones available, thinking that it will be useful for other people.

1. Browse existing *TestElement*s to determine if it or something similar is already available.
2. Provide data files for the new *TestElement* (see Sect. 6.1).
   (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 *TestElement*, link to that.
3. Fill in a form with basic information (described in Sect. 6.1). This info helps to categorize the *TestElement* in the database, and it will be passed along to future users who see the *TestElement*.
4. Upload *Result*s 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 *TestElement*. (If the new *TestElement* is better than an existing one, include a deprecation request?)

4.2.3 *Contribute a new performance result*

This could be part of the workflow in contributing a new *Method* or *TestElement*. But it could also be done separately. In that case, find combinations of *Method* + *TestElement* 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 *TestElement*s in a *ReferenceSet*. Alter the *TestElement*s and/or *ReferenceSet*s in a *BenchmarkSet*.

4.3.2 *Approve an addition from a Contributor*

Check the new *Method* or *TestElement* 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 *TestElement*?

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

This section is rough notes about the web interface a user will work with. Sketches and specific layout comments are in Sect. 8.

5.1 Drill down to results of interest

Beginning of workflow in Sect. 4.1.1.

1. Select the desired *Task*. We will focus on 'state-dependent diversification' with 'discrete traits' and probably 'hypothesis testing,' but eventually there will be others.
2. Select the desired *Method*s. Choose by model/technique (e.g., bisse, sister clades) and/or statistics (e.g., AIC, Bayes factors). Need a nice interface for this, but not too complicated.
3. Select the desired *TestElement*s. A few options here:
   - Those in the *BenchmarkSet*.
   - Those in particular *ReferenceSet*s. Select from the list of *ReferenceSet*s.
   - Select based on attributes of *TestElement*s, including attributes of *Tree*s and *Traits*. Will require a nice interface.
   - All. Warn how many this is.

5.2 Report for a task

End of workflow in Sect. 4.1.1.

Original vision in Fig. 1. This is the single most important part of the user interface.

Interactive interface:

- Select tags to float *TestElement*s to top (within a *ReferenceSet*)

- Same, but for categories (i.e., the possible values within a column)
- Click arrow at top of column to sort on its categories
- Click arrow to fold away a *ReferenceSet*
- Click something to fold away or resize columns?
- Apply a sequence of sort/filter actions. Each refines the previous one, rather than replacing it.
- Clear some/all the sort/filter actions
- Select *TestElement*s (rows) to hide
- Hide all non-floated/non-selected *TestElement*s
- Maybe color rows based on *Performance*
- Floating headers, remains visible when scrolling
- Table may be large. But show the whole thing, rather than 'paging' it. Be as space-efficient as possible.

Link to download CSV file (Sect. 7.3).

5.3 Report for a method

See Sect. 4.1.2.

Most functionality like Sect. 5.2. But in this case, there will be a column for each *Task* (instead of for each *Method*). Not sure how standard the task columns can be. Need to think about if all tasks/questions can be answered with one number or symbol.

5.4 See element

See details of a particular *TestElement* (Sect. 6.1).

Snazzy plot of trait on tree?

5.5 See method

See details of a particular *Method* (Sect. 6.7).

5.6 Download testing files

View to choose *TestElement*s, *ReferenceSet*s, *BenchmarkSet* (Sect. 7.1).

View to choose *Method*s (Sect. 7.2).

5.7 Contribute element

Form for a Contributor to provide the necessary information and files (Sect. 6.1).

5.8 Contribute method

Form for a Contributor to provide the necessary information and files (Sect. 6.7), and performance results.

5.9 Administrator

Need to consider extra tasks and Admin user would do.

6 Database Tables and Forms

This section defines the fields needed for each type of information (*TestElement*, *Tree*, etc.). Fields marked '[auto]' should be automatically populated when the contribution is saved. All other fields should be provided by the Contrib-

utor via an input form.

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

6.1 Elements

Each *TestElement* consists of one *Tree* (Sect. 6.2), optionally one *Trait* (Sect. 6.3), and some other information. There should be a form for a Contributor to create a new *TestElement*.

Unique ID Arbitrary, e.g., E-47295. [auto]

Date When this was created. [auto]

Contributor Who created this. [auto] Can be filled in automatically because a Contributor must be logged in to add anything new to the database (Sect. 3.2).

Name A brief description of the *TestElement*. Free text with a maximum of 50 characters. Might not be unique.

Comment Any other text the Contributor would like to provide about the *TestElement*. Free text. Formatting should be preserved.

Tree Link to one *Tree*. An existing *Tree* could be selected. Or, a new *Tree* could be created along with this *TestElement* (see Sect. 6.2).

Trait Link to one *Trait*, or empty. (Some Tasks may not require a Trait in each Element. But for our initial tasks, there will always be one.) An existing *Trait* could be selected. Or, a new *Trait* could be created along with this *TestElement* (see Sect. 6.3). We will need a way to check that the species names agree in a *Trait* and *Tree* that are part of the same *TestElement*.

Number of items Positive integer. To avoid ambiguity, the Contributor will provide this. But the value should be sanity-checked against the number of items in the *TestElement*'s *Tree* and *Trait*.

Reference set Link to one or more *ReferenceSet*s (Sect. 6.4) that this *TestElement* will belong to.

Correct answer What answer should be obtained. The *TestElement* will be associated with one or more *Task*s. (This is determined by the *ReferenceSet*s selected above, which each map to *Task*s.) For each *Task*, there is a correct answer that should be obtained from the *TestElement*. Ideally, the Contributor would be presented with a small table showing each relevant *Task*, with a box to provide the corresponding 'Correct answer'.

We will want to be able to find all the *TestElement*s to which a *Tree* or *Trait* belongs. From this, we can find all *Tree*s that use a *Trait*, and all *Traits* that go with a *Tree*.

6.2 Trees

It should only be possible to create a new *Tree* as part of a new *TestElement*. Thus, the form for these *Tree* fields should be part of the form for contributing an *TestElement*.

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

Unique ID Arbitrary, e.g., T-83247. [auto]

Date When this was created. [auto]

Contributor Who created this. [auto]

Name A brief description of the *Tree*. Free text with a maximum of 50 characters. Might not be unique.

Comment Any other text the Contributor would like to provide about the *Tree*. Free text. Formatting should be preserved.

Number of items The number of unique trees uploaded. For safety, have the user provide this but check it against the actual trees.

Number of tips The number of tips per tree. Could be a single number or a range. For safety, have the user provide this but check it against the actual trees.

Source A text file uploaded by the Contributor. Usually, this will be a script that generates the tree(s) or downloads them. If that is not possible, the file can contain notes explaining where the trees came from.

Tree files 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). The Contributor will provide one text file per tree, so we should allow multiple files or one zipped file to be uploaded.

Keywords A list of options, from which the Contributor will choose one or more. We will want to be able to search/filter *Tree*s based on these values. The exact keywords will be determined as we go along, but a starting point is: 'simulated', 'empirical', 'penalized likelihood'

6.3 Traits

It should only be possible to create a new *Trait* as part of a new *TestElement*. Thus, the form for these *Trait* fields should be part of the form for contributing an *TestElement*.

Each *Trait* object consists of at least one trait value per species. There could be multiple such sets in one *Trait* object, e.g., when simulating data so that each tree in a *Tree* goes with one trait set in a *Trait*. All the trait sets within a *Trait* have the same properties.

Unique ID Arbitrary, e.g., A-57387. [auto]

Date When this was created. [auto]

Contributor Who created this. [auto]

Name A brief description of the *Trait*. Free text with a maximum of 50 characters. Might not be unique.

Comment Any other text the Contributor would like to provide about the *Trait*. Free text. Formatting should be preserved.

Number of items The number of unique trait sets uploaded. For safety, have the user provide this but check it against the actual files.

Source A text file uploaded by the Contributor. Usually, this will be a script that generates the trait values or downloads them. It might be the same as the generating script for a corresponding *Tree*. If there is no script, the file can contain notes explaining how to get the trait values.

Traits files Each set of traits is simply a list of numbers, labeled by tip/species name. As for *Tree*s (Sect. 6.2), the state info could reside directly within the database or in a linked text file (e.g., CSV). The Contributor will provide one text file per set of traits (i.e., per tree), so we should allow multiple files or one zipped file to be uploaded.

Keywords A list of options, from which the Contributor will choose one or more. We will want to be able to search/filter *Traits* based on these values. The exact keywords will be determined as we go along, but a starting point is: 'empirical', 'simulated', 'continuous', 'discrete', 'binary', 'BM', 'OU', 'threshold', 'Mk', 'SSE', 'neutral', 'fast', 'slow'

6.4 Reference Set

Each *ReferenceSet* is a collection of *TestElement*s (perhaps hundreds), plus some other information.

Unique ID Arbitrary, e.g., R-43853. Auto-generated when created.

Name A short phrase that identifies the *ReferenceSet* to a human.

Description An explanation of what this *ReferenceSet* is designed to test.

Elements Link to *TestElement* (s) in the *ReferenceSet*. It should also be easy to obtain the total number of *TestElement*s.

Benchmarks Link to *BenchmarkSet* (s) that contain parts of this *ReferenceSet*, if any.

Methods Link to *Method* (s) for which this *ReferenceSet* is relevant.

History Notes from Contributors who have created or changed the *ReferenceSet*. (More than one column, but I'm not sure of the best format.)

(I'm not sure how much linking across tables is necessary. For example, I've included links to *BenchmarkSet* and *Method* here, but perhaps those connections could be obtained merely from the *BenchmarkSet* and *Method* tables themselves.)

6.5 Benchmark Set

Each *BenchmarkSet* is a collection of *TestElement*s (perhaps dozens), linked to one specific *Task*, plus some other information. These may be updated frequently, as new *TestElement*s are added and old ones are removed.

Unique ID There won't be many of them, so we could have more meaningful names. These could be constructed from the name of the *Task* and a number (in case we want to be trying out a few benchmark sets at once).

Name A short phrase that identifies the *BenchmarkSet* to a human.

Description An explanation of what this *BenchmarkSet* is designed to test.

Task Link to the *Task*.

Elements Link to *TestElement* (s) in the *BenchmarkSet*. It should also be easy to obtain the total number of *TestElement*s and their *ReferenceSet* membership.

Methods Link to *Method* (s) for which this *BenchmarkSet* is relevant.

Correct outcome What answer should be obtained by an effective *Method*, for a particular *Task*. This will be the same for all *TestElement*s in the *BenchmarkSet*.

History Notes from Contributors who have created or changed the *BenchmarkSet*. (More than one column, but I'm not sure of the best format.)

6.6 Task

The top layer of organization is the *Task*. There won't be many, and perhaps only one for awhile (i.e., state-dependent diversification).

Not discussed in the original proposal (Sect. 2.1) is the concept of grouping tasks or questions within them. For example, if the task is state-dependent diversification, there could be different sub-tasks for discrete-valued and continuous-valued traits. We might also want different questions for hypothesis testing (Is my trait associated with diversification shifts?) versus parameter estimation (How much higher are speciation rates for this state?). This latter level of detail is essential for reporting results simply (Sect. 6.8).

I think designing this layer of organization well will be quite important for PhyCoMB to be useful. But we may not know the best strategy until we see how it grows. So we'll need more discussions and probably some flexibility here. For the moment, let's assume that *Task* is defined as specifically as necessary. This suggests using only the following fields:

Unique ID There won't be many of them, so we could have meaningful names.

Name A short phrase that identifies the *Task* to a human.

6.7 Methods

Sects. 6.1–6.6 were all about the testing datasets themselves. Now we consider the analysis methods that the tests are designed to evaluate.

Unique ID Arbitrary, e.g., M-93925. [auto]

Date When this was created. [auto]

Contributor Who created this. [auto]

Name A brief description of the *Method*. Free text with a maximum of 50 characters. Might not be unique.

Comment Any other text the Contributor would like to provide about the *Method*. Free text. Formatting should be preserved.

Task Choose from the list of existing *Task*s that this *Method* is be used for. I'm pretty sure that each *Method* can only be used for a single *Task*, because the *Method* can only return a single value. The *Task* therefore determine the return value type (Boolean or numeric) of the *Method*.

Source The analysis script used to run the method. One or more text files.

Allied methods Note any existing *Method*s that are very closely related to this one. There may be too many *Method*s to present them all as a list. Perhaps the Contributor could start to type the UniqueID or Name and the options could auto-complete. (These links will need to be updated automatically, as well. For example, when I contribute Method 2 today I can note that it is allied with Method 1 that I created yesterday. But then the link from Method 1 to 2 should be created automatically.)

Model-based Choose 'yes' or 'no'.

Keywords A list of options, from which the Contributor will choose one or more. We will want to be able to search/filter *Method*s based on these values (as well as on 'Model-based'). The exact keywords will be determined as we go along, but a starting point is: 'Akaike information criterion', 'Bayes factor', 'bootstrap', 'model averaging', 'parameter estimates', 'posterior predictive', 'randomization test', 'semi-parametric', 'sign test'

Methods often come in groups, and I'm not sure how to handle this. Any thoughts or suggestions appreciated. For example, the same basic procedure could be tweaked in a few ways (in the algorithm, or the type of stats used), and testing would reveal which was best. Some possible approaches:

- Ignore this complication. Each *Method* is a stand-alone entity, and Contributors can just explain in the Comments if they want.
- Ignore this complication in the database structure, but maintain by hand a high-level summary of the available *Method*s. Could be tough if *Method*s are added frequently, though could ask Contributor for suggestions on how to update the summary accordingly.
- Collect enough meta-data (Columns, Tags, etc.) to generate a summary of relationships among *Method*s. Not clear that this can usefully be done algorithmically, though.
- Allow one *Method* to link to others. But then get into issues of how similar is similar enough to link (e.g., 'coded like' versus 'inspired by'). And then there would be lots of code reuse among scripts of different-but-related *Method*s. [note: this is the 'Allied methods' field specified above]

- Allow multiple analysis scripts and/or libraries of shared functions per *Method*. Seems likely to get messy, though, especially when summarizing results.
- Allow one explicit level of hierarchy in *Method*s. Contributor groups them together if they share code, but each is numbered separately and results are reported separately for each variant.

6.8 Results

When a *Method* is run on an *TestElement*, the output should be a clear answer to a *Task*. This is a *Result*. There will actually be a value for each Item in the *TestElement* (e.g., ten numbers if there are ten individual trees)—this is what the Contributor will provide. We'll want PhyCoMB to retain this level of detail so it is available for download. But in the PhyCoMB web interface, we will usually want to see a summary in which each Method-Element combination is reduced to a single value, probably the proportion of Items that yielded the correct answer (see the Outcome column in Fig. 4).

I think the best thing here will be for a Contributor to upload a CSV file with all the information needed to compute one or more *Result*s. The columns should be:

- Method (its Unique ID)
- Element (its Unique ID)
- Item (1, 2, 3, . . .)
- Value (TRUE/FALSE or a number)

That uploaded file will need to be checked before the contents are accepted. Validation should include:

- Each *Method* exists
- Each *TestElement* exists
- All the Items for that *TestElement* are included
- The Value is of appropriate type (Boolean or numeric) for the *Method* and *Task*
- The Value does not disagree with a value that was already reported

The form for uploading *Result*s can be minimal: just a box for CSV file(s).

The following fields should be automatically filled in when a new *Result* is created.

Date When this was created. [auto]

Contributor Who created this. [auto]

We will want to be able to connect each *TestElement* with the *Method*s that run on it. The *Result*s table provides this link.

One concern I have is that reducing each *Result* to a single number would cause analyses to be simple-minded rather than nuanced. On the other hand, we need to keep things simple enough that we can summarize across lots of tests and methods. Would be good to discuss this.

We will definitely need nice ways to summarize this huge table of results for users. These summaries are called *Reports* elsewhere in this document. But I'm not sure whether this would be done within the database structure or when generating views (e.g., Sects. 5.2 and 5.3). Some summaries we'll want:

- Overall performance of a *Method* on a *ReferenceSet* for a *Task*
- Overall performance of a *Method* on a *BenchmarkSet* for a *Task*
- Overall performance of a *Method* at a *Task*
- Performance of all *Method*s on an *TestElement*
- Performance of all *Method*s on an *ReferenceSet*
- Performance of all *Method*s on an *BenchmarkSet*

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 *TestElement* is downloaded, the user should receive:

- Information about it (Sect. 6.1), written in an auto-generated text file.
- *Tree* files and/or generating script and information (Sect. 6.2).
- If applicable, *Trait* files and/or generating script and information (Sect. 6.3).

These files should all have sensible names.

When multiple *TestElement*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 *TestElement*) should be included so it's easy to see which *TestElement*s have which attributes (columns in *Tree*, membership in *ReferenceSet*, etc.).

If an entire *ReferenceSet* is requested for download, each *TestElement* 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 *TestElement*, 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* (Sect. 5.2 or Sect. 5.3). Either the full report could be requested, or only to include those rows (*TestElement*s) and columns that are visible after interacting with the report view.

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

8 INTERFACE

This section has some examples of what the web user interface could look like. At the time of writing, the numbers in these mock-ups agree with the content of `practice_data/`.

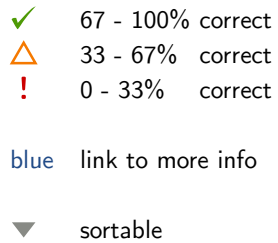

Figure 2: Graphical elements of the user interface. (i) Performance is reported as the proportion of items on which the correct result was obtained. The checkmark, triangle, and exclamation marks are quick visual indicators of whether results were good, cautionary, or bad. (ii) Text in a blue font is a link to a page that contains more information. (iii) Arrows in the header of a table indicate that the table can be sorted by that column. Sorting should be iterative, e.g., sort first on the contents of one column (click its arrow), and then within each of those categories sort based on a second column (shift-click its arrow).

8.1 Overview of how all methods perform

The highest-level display of results summarizes the performance of each *Method* on each *BenchmarkSet* for a given *Task* (Fig. 3). The goal is give a quick visual summary of which *Method*s do well, and which *BenchmarkSet*s are challenging.

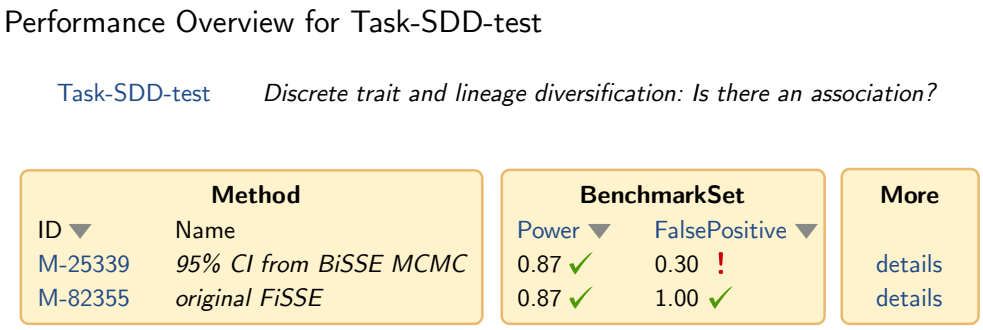

Figure 3: Performance of all *Method*s for a *Task*. Each number is the average of the *Method*'s success for each *TestElement* in that *BenchmarkSet*. The 'details' link leads to a view like Fig. 4. The links from each model (in the 'ID' column), each *BenchmarkSet* ('Power', 'FalsePositive'), and the *Task* ('Task-SDD-test') each lead to a page displaying all the information for that component (details, scripts, contributor information, etc.).

8.2 Performance of one method

Focusing on a single *Method*, we can see in more detail how it performs across all the *TestElement*s that comprise the *BenchmarkSet*s (Fig. 4).

Performance of Method M-25339 for Task-SDD-test

M-25339	*95% CI from BiSSE MCMC*
Task-SDD-test	*Discrete trait and lineage diversification: Is there an association?*
BenchmarkSets	FalsePositive, Power

☐ Show Tree information
☐ Show Trait information

BenchmarkSet	Element ID	Element Name	Performance Correct	Performance Outcome
FalsePositive	E-78458	*neutral threshold on empirical tree*	no	0.20 ! 2/10
FalsePositive	E-29543	*fast neutral binary on empirical tree*	no	0.40 △ 4/10
Power	E-17888	*bisse simulated trait*	yes	0.87 ✓ 13/15

☑ Show Tree information ☑ Show Trait information

Tree ID	Tree Name	Tree NumTips	Tree Source	Trait ID	Trait Name	Trait Type
T-83768	*corals*	358	empirical	A-23904	*slow neutral*	threshold
T-83768	*corals*	358	empirical	A-03875	*fast neutral*	discrete
T-13983	*bisse sim*	36-1221	simulated	A-11463	*bisse sim real*	discrete

Figure 4: Performance of one *Method* for all relevant *BenchmarkSet*s. Each number is the proportion of datasets (tree-trait combinations) in an *TestElement* on which the *Method* gave the 'Correct' answer. (This column header might be confusing, but I mean that the correct answer to the *Task*'s question is 'no' for the first *TestElement*, etc.) The numerical links ('2/10' etc.) lead to a more detailed report of the results. The checkboxes allow the user to specify whether more information about the *Tree* and/or *Trait* in each *TestElement* should be displayed. If a box is checked, further columns are displayed in the table (to the right of 'Performance'), as illustrated below the dotted line.