ARCHIPELAGO

Jeet Sukumaran

March 17, 2016

Contents

1 Introduction

"*Archipelago*" is the name of a generative phylogeny-based model that simultaneously incorporates the diversification processes of speciation and extinction nad the biogeographical processes of area gain ("dispersal") and area loss ("extirpation"), with these processes being differentially regulated by ecological or other traits that are themselves co-evolving on the phylogeny. The theory and background to this model and its usage is described in the following paper:

This software project, `archipelago`, presents a suite of programs to generate and analyze data under the *Archipelago* model. The primary objective of the analysis is *model selection*: i.e., statistically identifying the model that generated a particular set of data. The suite of programs constituting `archipelago` is, thus, a computational biogeographical model selection analysis package that exploits the power and flexibility of the *Archipelago* model to allow you to ask and answer historical biogeographical questions of a nature and complexity that are not possible under any other approach. In particular, instead of asking questions about ancestral area patterns, you can ask questions about processes, and how some processes (e.g., ecology) affect other processes (e.g., dispersal or speciation).

2 Installation

2.1 Mandatory Pre-requisites

- Python 2.7 or higher
- DendroPy Phylogenetic Computing Library, version 4 or above (http://dendropy.org).
- R (http://www.r-project.org/)
- The following R package:
  - adegenet

2.2 Optional Pre-requisites

To "calibrate" the simulation biogeographical submodel, you need to estimate the dispersal and extinction rate under the DEC or a DEC-like model, while to calibrate the simulation trait model, you need to estimate the trait evolution rate. `archipelago` provides a script that will "profile" your empirical tree for you, estimating these rates.

To take advantage of this, you need at least one of the following installed to estimate the dispersal and extinction rate under the DEC model:

- lagrange
- BioGeoBears (R package)

However, you can also estimate these parameters yourself using BayArea or RevBayes. In fact, I would *highly* recommend that you do the latter if you have a large number of geographical areas (e.g. more than 8), or you wish to incorporate uncertainty in the parameter estimates.

Similarly, the `archipelago` uses the R package "GEIGER" to estimate the trait evolution rate. You can, of course, use any other method that you prefer.

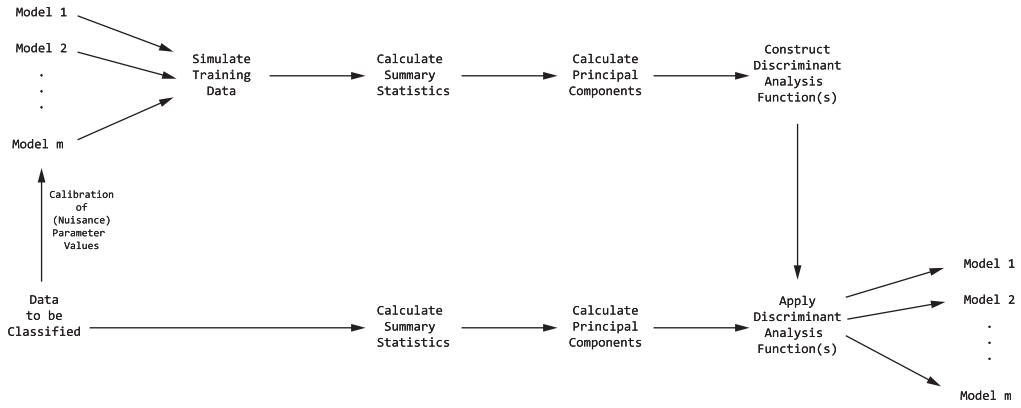
2.3 Installation from Source

Run the following in the top-level directory of the project:

```
python setup.py install
```

3 Usage

3.1 Overview of Workflow



Let D be the *target* data, consisting of an ultrametric phylogeny with each of the tips associated with a geographic range (presence/absence over one or more areas) as well as with set of trait states. Let $\mathbf{M} = \{M_1, M_2, ...M_m\}$ be a set of k models that we are interested in studying. Let $S(\cdot)$ be a function that takes a set of data returns a set of summary statistics. Our analytical objective is, for each $M_i, M_i \in \mathbf{M}$, estimate the probability that it generated the target data D, relative to all the other models in $\mathbf{M}$. The operational procedure consists of the following steps:

1. **Simulation of Training Data**: For each $M_i, M_i \in \mathbf{M}$, simulate n replicates of data, D^{*i}. We label each replicate of this data with the name of the model that generated it (e.g., "Model i"). The set of all data so generated constitutes the training data, $\mathbf{D}^*$.

2. **Calculation of Summary Statistics**: Calculate summary statistics on the training data to yield, $S(\mathbf{D}^*)$, and summary statistics on the target data to yield $S(D)$.

3. **Classification of Target Data**: Construct a Discriminant Analysis (DA) function on principal components (PC) calculated on the training data set summary statistics, $S(\mathbf{D}^*)$; apply the discriminant analysis function to principal components calculated on the summary statistics of the target data, $S(D)$.

3.2 Simulation of Training Data

We use the program, "`archipelago-simulate.py`", to simulate the data under each model in the set of competing models. The simulation program is run separately for each of the models, with multiple independent replicates in each run. Full details on this program invocation can be found by running:

```
archipelago-simulate.py --help
```

A typical invocation of the program will have the following arguments:

- **"MODEL-FILE"**
  A JSON or Python file containing nothing but a dictionary specifying the full model under which to simulate data. This is described in detail in the Model Specification section.

- **"`-n`" or "`--nreps`"**
  The number of independent replicates to simulate under this model.

- **"`-o`" or "`--output-prefix`"**
  The path and filename stem where the results will be saved.

For example,

```
archipelago-simulate.py -n 100 -o results/m1 model1.json
```

will run 100 replicates under the model specified in the file "model1.json" and all output will saved to files in the "results/" subdirectory with filenames beginning with "m1".

The simulation program produces the following output:

- `<OUTPUT-PREFIX>.focal-areas.trees`
  A collection of phylogenies in NEWICK format, where each phylogeny represents the result from a single simulation replicate. The tips of these trees represent lineages that occur in at least one of the focal areas (in contrast to the phylogenies in the "`all-areas.trees`" file, where *all* extant lineages are represented, regardless of where they occur). The distribution and trait states of the tip lineages are encoded in the tip labels. This is the data that is of primary analytical interest, and will be passed to the summary statistics calculation program, `archipelago-summarize.py`.

- `<OUTPUT-PREFIX>.all-areas.trees`
  A collection of phylogenies in NEWICK format, where each phylogeny represents the result from a single simulation replicate. The tips of these trees represent *all* extant lineages at the end of the simulation, regardless of whether or they occur in the focal areas. The distribution and trait states of the tip lineages are encoded in the tip labels.

- `<OUTPUT-PREFIX>.log`
  A detailed log of the run.

- `<OUTPUT-PREFIX>.model.json`
  A description of the model that the run executed. This can be used to confirm that the model used corresponded to the model that was input. In addition, for default settings or values may have been used aspects or parameters that were not specified in the input model file, and these are recorded here.

3.2.1 Model Specification

We specify the model under which we simulate data using `archipelago-simulate.py` by a *model file*. This is a JSON file providing a single dictionary as its only element. An example of a basic model specification is:

```
{
  model_id: "samplemodel1",
  "areas": [
    {
      "label": "a1", "is_supplemental": false, "relative_diversity": 1.0,
      "area_connection_weights": [0.0, 1.0, 1.0, 1.0, 1.0]
    },
    {
      "label": "a2", "is_supplemental": false, "relative_diversity": 1.0,
```

```
      "area_connection_weights": [1.0, 0.0, 1.0, 1.0, 1.0]
    },
    {
      "label": "a3", "is_supplemental": false, "relative_diversity": 1.0,
      "area_connection_weights": [1.0, 1.0, 0.0, 1.0, 1.0]
    },
    {
      "label": "a4", "is_supplemental": false, "relative_diversity": 1.0,
      "area_connection_weights": [1.0, 1.0, 1.0, 0.0, 1.0]
    },
    {
      "label": "s1", "is_supplemental": true, "relative_diversity": 1.0,
      "area_connection_weights": [1.0, 1.0, 1.0, 1.0, 0.0]
    }
],
 "traits": [
    {
      "label": "Trait1", "nstates": 3, "transition_rate": 0.01,
      "transition_weights": [
          [0.0, 0.5, 0.5],
          [0.5, 0.0, 0.5],
          [0.5, 0.5, 0.0]
      ]
    },
    {
      "label": "Trait2", "nstates": 2, "transition_rate": 0.01,
      "transition_weights": [
          [0.0, 1.0],
          [1.0, 0.0]
      ]
    }
],
 "diversification": {
    "mean_birth_rate": 0.1,
    "lineage_birth_weight": {
      "definition_type": "lambda_definition",
      "definition": "lambda lineage: 1.0",
      "description": "fixed: 1.0"
    },
    "mean_death_rate": 0.1,
    "lineage_death_weight": {
      "definition_type": "lambda_definition",
      "definition": "lambda lineage: 1.0",
      "description": "fixed: 1.0"
    }
 },
 "anagenetic_range_evolution": {
    "global_area_gain_rate": 0.2,
    "lineage_area_gain_weight": {
```

```
      "definition_type": "lambda_definition",
      "definition": "lambda lineage: 1.0",
      "description": "fixed: 1.0"
    },
    "mean_area_loss_rate": 0.1,
    "lineage_area_loss_weight": {
      "definition_type": "lambda_definition",
      "definition": "lambda lineage: 1.0",
      "description": "fixed: 1.0"
    }
  },
  "cladogenetic_range_evolution": {
    "sympatric_subset_speciation_weight": 1.0,
    "single_area_vicariance_speciation_weight": 1.0,
    "widespread_vicariance_speciation_weight": 1.0,
    "founder_event_speciation_weight": 0.0
  },
  "termination_conditions": {
    "target_focal_area_lineages": 50,
    "gsa_termination_focal_area_lineages": null,
    "max_time": null
  }
}
```

includes/samplemodel1.json

The model dictionary consists of the following elements:

"model_id"

This key takes an arbitrary string value that identifies the category or class of model being run. It will serve as the identifier of all data generated under this model. Each tree generated under this model will have the metadata item, "`model_id`", with this value associated with it. Each set of summary statistics calculated on each tree will have an additional field, "`model_id`" added to it to document the generating model by the value given to this field. The classification programn "`archipelago-classify.py`", will expect the training data summary statistics to have a column with this name, and it will use the values of this column to determine the origin or source of the data.

"areas"

This key defines the geographical template. Its corresponding value is a list of dictionaries with the following keys, with each dictionary defining a distinct area:

"label":
A unique **string** value giving the name or identifier for the area.

"is_supplemental":
A **boolean** (`true` or `false`) specifying whether or not the area is a focal area or a supplemental area. Focal areas are areas from which the lineages are sampled, while supplemental areas are area that interact with the focal areas ecologically, evolutionarily, and biogeographically, but are not sampled in the target data. For example, when analyzing a group of Western Pacific islands, we may consider the island groups to be focal areas, but Australia and Papua New Guinea may be considered supplemental areas, because our target data only spans lineages that have representatives in the focal areas, though there is no doubt that the evolutionary history of those lineages includes Australia and Papua New Guinea.

"area_connection_weights":
A list of positive real values which provides the dispersal weights from the area being defined to all other areas, specified in the order when they are defined. Note that this includes the connection weight to the same area as well, and this is *must* be set to 0.0. If not specified, by default it is assumed that all areas are equally connected (i.e., connection weights default to 1.0, except for the self-connection weight, which is 0.0.).

"traits"
This key defines the traits. Its corresponding value is a list of dictionaries with the following keys, with each dictionary defining a distinct trait:

"label":
A unique **string** value giving the name or identifier for the trait.

"nstates":
A **non-zero positive integer** specifying the number of distinct states that the trait has.

"transition_rate":
A **positive real** specifying the mean rate of change per unit time for the trait.

"transition_weights": (optional)
A list of lists specifying the relative weights (as **positive real** values) for each state transition. If not given, equal rates are assumed for all transitions.

"diversification"

This key defines the diversification processes (speciation and extinction). It consists of a dictionary with the following two keys:

"mean_birth_rate":

The birth rate of the branching processes under a birth-death model: i.e., the mean per-lineage rate of speciation. This will be modified for each lineage by the lineage birth weight function specified below, but the simulation will maintain this mean across all lineages.

"lineage_birth_weight":

A *lineage-specific weight function definition* (see below) specifying the weight of the probability of speciation of this lineage relative to the others. This will be normalized across the phylogeny to maintain the birth rate given by *mean_birth_rate*.

"mean_death_rate":

The death rate of the branching processes under a death-death model: i.e., the mean per-lineage rate of extinction. This will be modified for each lineage by the lineage death weight function specified below, but the simulation will maintain this mean across all lineages.

"lineage_death_weight":

A *lineage-specific weight function definition* (see below) specifying the weight of the probability of extinction of this lineage (i.e., the death weight in a birth-death branching process submodel, *not* the "extinction" or area loss weight of the DEC biogeographical submodel) relative to the others. This will be normalized across the phylogeny to maintain the death rate given by *mean_death_rate*.

"anagenetic_range_evolution"

This key defines the anagenetic range evolution processes (area gain and area loss). It consists of a dictionary with the following two keys:

"global_area_gain_rate":

This corresponds to the area gain rate parameter in the BayArea model the d or "global dispersal" parameter in the DEC model.

"lineage_area_gain_weight":

A *lineage-specific weight function definition* (see below) specifying the relative weight at which a lineage gains an area in its range relative to the other lineages. This will be normalized across the phylogeny to maintain the global area gain rate given by "*global_area_gain_rate*".

"mean_area_loss_rate":
This corresponds to the area loss rate in the BayArea model the e or "extinction" (really, extirpation) parameter in the DEC model.

"lineage_area_loss_weight":
A *lineage-specific weight function definition* (see below) specifying the relative weight at which a lineage loses an area in its range relative to the other lineages. This will be normalized across the phylogeny to maintain the mean area loss rate given by "*mean_area_loss_rate*".

"cladogenetic_range_evolution"
This key defines the relative weights of the cladogenetic range evolution speciation modes. It consists of a dictionary with the following keys, with the value of each key being the weight of that mode relative to all other modes:

"sympatric_subset_speciation_weight":
Default value: 1.0.

"single_area_vicariance_speciation_weight":
Default value: 1.0.

"widespread_vicariance_speciation_weight":
Default value: 1.0.

"founder_event_speciation_weight":
Default value: 0.0.

"termination_conditions"
This key defines the conditions which determine when the simulation should end. Its value is a dictionary consisting of *one* of the following keys:

"target_focal_area_lineages":
A non-zero positive integer that specifies the minimum number of extant lineages that occur across all focal areas. Once this number is reached or exceeded, the simulation terminates. Typically, if this mode of termination is used, this will be set to the number of tips of the target data phylogeny.

"max_time":
A positive real number that specifies the maximum time the simulation should run. Typically, if this mode of termination is used, this will be set to the root age of the target data phylogeny.

3.2.2 Lineage-Specific Weight Function

A number of process rates (speciation, extinction, area gain, area loss) are modified by a "lineage-specific weight function". This allows the rate of the process to vary depending on the trait state or other attributes of the lineage. For each process, the weight function can be specified in one of four ways: a fixed value, a (Python) lambda expression, a trait state mapping, or as a Python function object.

3.2.2.1 Fixed-Value Lineage-Specific Weight Function

With a fixed-value lineage-specific rate function, the rate of the process is the same, fixed, constant value regardless of lineage. That is, all lineages unconditionally have the same relative rate for the process throughout their lifetimes in the simulation. A fixed-value lineage-specific rate function is specified by providing a dictionary like the following as the value for the process key:

```
{
    "definition_type": "fixed_value",
    "definition": 1.00,
}
```

The value assigned to "definition" does not really matter, as it will be normalized across all lineages and multipled by the mean or global rate for the process: the key point here is that all lineages get the same weight.

3.2.2.2 Trait State Mapping Lineage-Specific Weight Function

Here, you provide a list of weight values for that processes parameter, with a one-to-one correspondence to the states of an trait defined previously. For example, if you have defined a trait with the label "color" and three states:

```
{"label": "color", "nstates": 3, "transition_rate": 0.01}
```

you could define a process that takes on the is weighted by 0.0 if the "color" trait of a lineage was in state 0, 0.5 if the trait was in state 1, and 1.0, if the trait was in state 2 by specifying:

```
{
    "definition_type": "trait_state_index_map:color",
    "definition": [0.0, 0.5, 1.0],
}
```

These weight represent the rate of the process for a lineage *relative* to others based on the trait state. The overall rate of the process across the phylogeny remains the same; it is just "distributed" across lineages differentially based on the state of the "color" trait associated with the lineage.

3.2.2.3 Lambda-Expression Weight Function

The weight function is given as a Python lambda function definition. The lambda function should take a single argument, an object of the "Lineage" class, and return a real value representing the weight with which to modify the process for this lineage, relative to other lineages, at this particular point in time. For example, the following replicates the above trait state mapping:

```
{
    "definition_type": "lambda_definition",
    "definition": "lambda lineage: 0.0 if lineage.traits_vector[0] == '0' else
     (0.5 if lineage.traits_vector[0] == '1' else 1.0)"
}
```

Note that trait *types* are 0-based numeric indexes, while trait states are character values, "0", "1", "2", etc. `archipelago` internally uses character values to specify trait states as this allows for special states such as "NA" or "null". Again, these weights represent the rate of the process for a lineage *relative* to others based on the trait state, and the overall rate of the process across the phylogeny remains the same.

As another example, the following inspects the distribution vector of a lineage and returns a weight of 1.0 if the first area is occupied and 0.5 if not:

```
{
    "definition_type": "lambda_definition",
    "definition": "lambda lineage: 1.0 if lineage.distribution[0] == 1 else
    0.5"
}
```

In constrast to trait states, absence/presence in an (0-based indexed) area is indicated by *numeric* values, 0 and 1, respectievly.

3.3 Calculation of Summary Statistics

3.4 Classification of Target Data

3.5 Profiling the Target Data

3.6 Encoding the Target Data

To calculate the summary statistics or profile the target data using `archipelago`, we need to communicate the trait and geographical data associated with the tips of the the target phylogeny to the program. This is achieved by encoding the information in the labels of the phylogeny. The tip label for each lineage consists of *three* components separated by a carat ("^") character:

```
s68^1.2.0^0011
```

The first component ("s68" in the above example) is an arbitrary species index that uniquely identifies each lineage.

The second component ("1.2.0" in the above example) is the vector of trait states for the species. The states of each trait for the lineage is represented by an integer index separated by a period. In this example, there are three trait types, with the value of the first trait type "1", the second "2", and the third "0". Note that the first trait state is indexed by 0, not 1. That is, if there are "n" states, then the set of valid trait state indexes is $\{0, 1, 2, ..., n-1\}$. All lineages in the system must have the same number of trait types (three, in this example), and the trait state values are assumed to be given in sequential order.

The third component ("0011" in the above example) is the geographical presence/absence vector for the lineage, where "0" indicates the absence of a lineage from the area, and "1" indicates the presence. In this example, the lineage is absent from the first and second areas, but present in the third and fourth. All lineages must have their presences or absences specified in all areas.

Consider a system given by the following phylogeny:

```
[&R] ((A:1,B:1):3,(C:3,(D:2,E:2):1):1);
```

where the traits are represented by the following character matrix:

```
A   001
B   011
C   201
```

```
D   100
E   211
```

and the distribution over six areas is represented by the following incidence matrix:

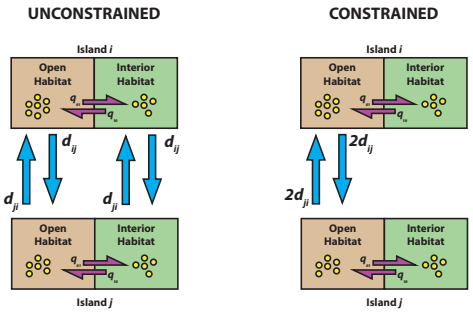
```
A   110001
B   010011
C   101101
D   001100
E   011111
```

We would then encode this information in the phylogeny as follows:

```
[&R] ((A^0.0.1^110001:1,B^0.1.1^010011:1):3,(C^2.0.1^101101:3,(D
^1.0.0^001100:2,E^2.1.1^011111:2):1):1);
```

4 Walk-Through of a Basic Analysis

4.1 Analytical Objectives



As an example of a very basic analysis, let us consider testing whether or not dispersal is constrained by habitat association of a lineage (Figure). We propose two different models, each corresponding to a different dispersal regime. In the "unconstrained" model, lineages are free to disperse between islands regardless of their habitat affinities. In the "constrained" model, ony lineages associated with the disturbed or open habitats can disperse between islands, while lineages associated with interior habitats are not able to disperse. The habitat affinities of the lineages themselves are modeled as a discrete binary trait that evolves stochasitcally over the phylogeny. That is, each lineage has a habitat trait, which can either be in the "disturbed" or "interior" state, and the state of this trait determines whether or not the lineage is capable of dispersing in the "constrained" model. Thus, in the unconstrained model, lineages disperse between interior habitats on different islands directly. In the constrained model, on the other hand, lineages cannot disperse between interior habitats directly: they have to move into a disturbed habitat association in ecological space over evolutionary time, at which phase they are capable of dispersing into new islands, and then reinvade the interior habitats on those new islands by, again, moving into them in ecological space over evolutionary time.

4.2 The Target Data

We will analyze the data set discussed in Sukumaran *et al.* 2015. This is a data set of birds in Wallacea, originally presented in Carstensen *et al.* 2013. We will be focussing on a subset of 365 species for which sufficient ecological data is available that allows us to categorize them into either "open" (disturbed) or "undisturbed" habitat species. The data, model files, etc. for this analysis can be found in the "`examples/wallacea_bird_dispersal_by_habitat/`" subdirectory of the project directory. The subdirectory "`original-data`" has the full phylogeny of 549 species ("`master_tree.nexus`"), from the Jetz *et al.* (????) paper as well as a comma-separated value formatted file with the geographical, ecological, and other information from the Carstensen *et al.* 2013 paper.

4.3 Simulating the Training Data

We will simulate data under two classes of models: "unconstrained" and "constrained". In this analysis, we will be using the basic DEC biogeography model as our bigeography (sub)model. We will need to specify the following rates:

- The speciation or birth rate

  In the DEC model as implemented by lagrange and BioGeoBEARS, speciation is not an explicit process: the model is conditioned on the speciation event timings and structure given by the input phylogeny, which is (typically) taken as truth. However, in *Archipelago*, we explicitly model the diversification process and sample phylogenies as part of the training data. While *Archipelago* supports a full birth-death model, we will not be modeling lineage extinction directly under this model. Instead, following the DEC model, we will model lineage extinction as part of the biogeographical submodel, with a lineage going extinct when its range is reduced to the null set, i.e. when it has lost all areas from its range. The speciation process is a nuisance process with respect to our study question, and both "unconstrained" and "constrained" classes of models will be set to the same fixed speciation rates. We will estimate the speciation rate under a pure-birth Yule model using the empirical target phylogeny.

- The trait transition rate

We use a simple equal-rates (ER) trait transition submodel. As with the diversification submodel, the trait evolution process is nuisance process with respect to our study question, and both "unconstrained" and "constrained" classes of models will be set to the same fixed trait transition rate, as estimated from the empirical target data.

- The global rate of area gain

  This corresponds to the "dispersal" or "d" parameter in the DEC model. We will estimate this rate based on the empirical target data under the basic DEC model, using lagrange, BioGeoBEARS, BayArea, or RevBayes. In the "unconstrained" class of model, this rate will be used by all lineage equally. In the "constrained" class of model, we will use a trait-state mapping function which will return a modifier weight of 0 if the lineage habitat trait is in "interior" state and a modifier weight of 1.0 the lineage trait is "disturbed" state. This means that lineages associated with the "interior" habitat will not disperse and colonize any new areas, while lineages associated with the "disturbed" state will. The actual rate of the area gain process for each lineage associated with the "disturbed" habitat state will vary, but will fulfill the following constraints:

  1. All lineages with the "disturbed" habitat state will have the same rate of area gain at any point in time
  2. The phylogeny-wide area gain rate will be the "global rate of area gain" specified.

  This means, that, at typically stationary distribution under an equal-rates transition model, where on the average half the lineage are in the "disturbed" state, the actual rate of area gain per "disturbed"-area lineage will be approximately twice the global area gai rate to compensate for the fact that half the lineages in the phylogeny are not dispersing.

- The global rate of area loss

  This corresponds to the "extinction" (extirpation) or "e" parameter in the DEC model. We will estimate this rate based on the empirical target data under the basic DEC model, using lagrange, BioGeoBEARS, BayArea, or RevBayes. The rate of area loss is the same under both the "unconstrained" and "constrained" models, and thus the estimated rate will be used as a fixed rate for this process under both these models.

4.3.1 Estimating the Rate Parameters from the Empirical Data to Calibrate the Simulations

Setting the parameters for the various processes (diversification, area gain, area loss, etc.) is known as *calibrating* the model. As noted above, this is typically done by setting the rates to estimates based on the target data. The `archipelago` package provides a program, "`archipelago-profile.py`" that conveniently does this for you. It requires that the target phylogeny labels have trait and geographical data encoded. The subdirectory "`target-data`" of the Wallacean example directory has the original phylogeny pruned down to the 365 species that are the focus of this study, with trait and geographical data encoded in the labels. We can run the profiler program on this phylogeny to yield the training data process rate parameters:

```
archipelago-profile.py --estimate-dec-lagrange target-data/wallacea.newick
```

if we have (the C++) `lagrange` installed, or, otherwise:

```
archipelago-profile.py --estimate-dec-biogeobears target-data/wallacea.newick
```

if we want to estimate the DEC model parameters using *BioGeoBEARS*.

This program will provide a "profile" of this phylogeny (provide MLE's of speciation, trait transition, area loss, and area gain rates). The result is a comma-separated value (CSV) formatted-file with one entry per tree passed to it. As here we are only using a single tree, there is only one data row. The results can be seen in the file "`wallacea.profile.csv`" which includes the following information:

Field	Value
num.tips	365
root.age	106.998428
pure.birth.rate	0.05482309568853121
trait.1.est.transition.rate	0.062247
area.est.transition.rate	2.396912
biogeobears.dec.dispersal.rate	0.018156246
biogeobears.dec.extinction.rate	0.002645152

4.3.2 Setting up the Training Data Models

We will create two models to generate the training data, "unconstrained" and "constrained". The files corresponding to these models can be found in the

"`examples/wallacea_bird_dispersal_by_habitat/training-data-models`" subdirectory of the project: "`unconstrained.json`" and "`constrained.json`", respectively.

The first entry in each model file is the identifier for each of the models:

`unconstrained.json`:

```
{
"model_id": "unconstrained",
}
```

`constrained.json`:

```
{
"model_id": "constrained",
}
```

These identifiers are crucial to allow the training data items to be assigned to the correct generating model, which itself is crucial for the proper construction of the discriminant analysis functions. The model identifiers will be assigned to the each tree generated by the simulations under each model, and will be propagated through rest of the pipeline by other programs.

Both models share the same geographical template, consisting of four focal areas and two supplemental areas, and are coded identically. The four focal areas represent the four island "modules" in the original data: Banda Sea, Lesser Sundas, Maluku, and Sulawesi. The two supplemental areas represent the two continental sources: Sundaland (Asia) and the Sahul Shelf (Australia and Papua New Guinea).

```
"areas": [
    { "is_supplemental": true, "label": "b1" },
    { "is_supplemental": true, "label": "b2" },
    { "is_supplemental": false, "label": "a1" },
    { "is_supplemental": false, "label": "a2" },
    { "is_supplemental": false, "label": "a3" },
    { "is_supplemental": false, "label": "a4" }
],
```

Both models also share the same trait regime: a single two-state trait, representing the habitat associated with each lineage. We use the trait transition rate estimated under an equal-rates model on the original data as the training data simulation trait transition rate.

```
"traits": [
    { "label": habitat, "nstates": 2, "transition_rate": 0.062247}
],
```

Both models also share the same diversification regime, and we use the birth rate estimated from the original data as the fixed, constant birth rate for both models. We also set the death rate to a fixed, constant rate of 0.0 for both models. (The Archipelago model distinguishes between the death rate of the

birth-death branching process, which is sometimes referred to as the "extinction" rate, and the rate of area loss, which is also called the "extinction" rate in the original DEC model description and related literature. Here, following the DEC model, we do not explicitly model extinction of lineages, but instead model extinction implicitly through modeling of the rate of area loss.)

```
"diversification": {
    "mean_birth_rate": 0.05486829846551938,
    "mean_death_rate": 0.0
},
```

We do not specify lineage-specific weight functions, and these default to equal and fixed weights across all lineages, equivalent to:

```
    "lineage_birth_rate": {
        "definition_type": "fixed_value",
        "definition": 1.0
    },
    "lineage_death_rate": {
        "definition_type": "fixed_value",
        "definition": 1.0
    },
```

The anagenetic rate of area gain is where the models differ. Both models actually share the same global area gain rate as estimated on the original data:

```
"anagenetic_range_evolution": {
    "global_area_gain_rate": 0.018156246,
    .
    .
    ,
```

But the overall rate is distributed across lineages differently in the two models based on the state of the habitat trait of the lineages.

In the "unconstrained" model, all lineages share the same rate:

```
"anagenetic_range_evolution": {
    "global_area_gain_rate": 0.018156246,
    "lineage_area_gain_rate": {
        "definition_type": "fixed_value",
        "definition": 1.0
    },
    .
    .
    .
```

However, in the "constrained" model, we only allow lineages that are associated with a state index of "0" (which we are arbitrarily assigning to mean the "open" or "disturbed" state) are allowed to disperse:

```
"anagenetic_range_evolution": {
    "global_area_gain_rate": 0.018156246,
    "lineage_area_gain_rate": {
        "definition_type": "trait_state_index_map:habitat",
        "definition": [ 1.0, 0.0 ]
    },
    .
    .
    .
```

We use a trait state mapping function to specify this regime, where lineages that are associated with a state index of "1" are prohibited from dispersing by assigning them a rate modifier weight of 0.0. Here, we identify the trait by the label that we assigned to it in the trait definition ("habitat") in the "definition_type" field, and for the "definition" field we provide a list of values, with each element in the list specifying the rate rate for the state with the corresponding index, so that the first rate is associated with the first trait state, the second rate is associated with the second trait state, and so on.

4.3.3 Running the Simulations

We use the program "`archipelago-simulate.py`" to simulate the training data. We generate 200 replicates under each of the models, saving the results to appropriately named output locations:

```
archipelago-simulate.py --nreps 200 --output-prefix unconstrained_training
    unconstrained.json
archipelago-simulate.py --nreps 200 --output-prefix constrained_training
    unconstrained.json
```

The above commands will generate two sets of output files, one beginning with the prefix "unconstrained_training" and the other beginning with the prefix "constrained_training". See the output file description section for details on the files produced. Here, we are interested in the files with the extension "`.focal-areas.trees`": these are the phylogenies that we need for the next stage. The above commands can be run on multiple computers simulataneously, or launched on the a multi-core computers multiple times in parallel, with all the "`.focal-areas.trees`" collected and passed to the next stage: summary statistic calculation.

4.4 Calculation of Summary Statistics

We calculate summary statistics on both the training data as well as the original empirical data. We use the program "`archipelago-summarize.py`" to calculate the summary statistics. This program takes the path of one or more phylogenies as arguments and produces a comma-separated value file of summary statistics as output, with one row of summary statistics for each phylogeny passed as input. While not a mandatory argument, explicitly specifying the output filenames using the "`-o`" or "`--output-filepath`" option is a good idea; otherwise, output gets written to the standard output instead of being saved to a file. The input phylogenies must be in NEWICk format, and have trait and geographical data encoded in the labels. The simulation program "`archipelago-simulate.py`" does this naturally, and, in addition, tags the trees with a model identifier label in such a way that "`archipelago-summarize.py`" recognizes this and adds it to the CSV file in a way that allows the next program in the pipeline, "`archipelago-classify.py`", to use it. Note that we must calculate the summary statistics of the training data *separately* from the target data, and save the results of each to separate files.

To calculate the summary statistics for the training data we issue the following command (assuming that the simulations in the previous step wrote their output to files with "`unconstrained_training`" and "`constrained_training`" prefixes:

```
archipelago-summarize.py --output-filepath summaries/training.summaries.csv
    unconstrained_training.focal-areas.tree constrained_training.focal-areas.
    tree
```

This will result in a file with the name "`training.summaries.csv`" in the `summaries` subdirectory that contains all the training data summary statistics for the next stage.

To calculate the summary statistics for the target data we issue the following command:

```
archipelago-summarize.py --output-filepath summaries/target.summary.csv target
    -data/wallacea.newick
```

This will result in a file with the name "`target.summary.csv`" in the `summaries` subdirectory that contains the target data summary statistics for the next stage.

4.5 Model Selection

The final stage of the analysis is where we attempt to classify the target data with respect to the generating model, i.e. model selection. We use the program "`archipelago-classify.py`" for this. This program takes, as a minimum, a path to a target data summary statistics file and one or more paths to training data summary statistics files. The first positional argument is take to be the target data summary statistics file while the remaining positional arguments are taken to be the training data summary statistics files. As above, it is a good idea to explicitly specify and output filepath using the "`-o`" option to save the output to a file. In addition, it is *highly* recommended to use the "`--describe-trained-model`" option to save details about the machine learning model used to classify the target data. We also need to tell the classifier how many principal components to use to build the discriminant analysis function. We will use the "`--optimize-npca`" to tell the classifier to use as the number of principal components that maximize the probability of correctly (re-)classifying the training data, with penalty factor of 0 (i.e., do not apply a penalty weight for each principal component used). So, for example, we run our the classification program on the data that we have generated above using the following command:

```
archipelago-classify.py --optimize-npca 0 -o result.csv --describe-trained-
    model dapc-details.txt summaries/target.summary.csv summaries/training.
    summaries.csv
```

This primary classifier output, written to "`result.csv`" in the above, is comma-separated value file which include the following fields:

assign
: The value of this field is the identifer (name or label) of the generating model that most probably generated the data.

posterior.`<MODEL_ID>`
: This is the relative weight in support of the model with the identifier "`<MODEL_ID>`".