# ARCHIPELAGO

Jeet Sukumaran

November 12, 2015

# 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 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 packages:
  - adegenet
  - picante
  - BioGeoBears
  - GEIGER

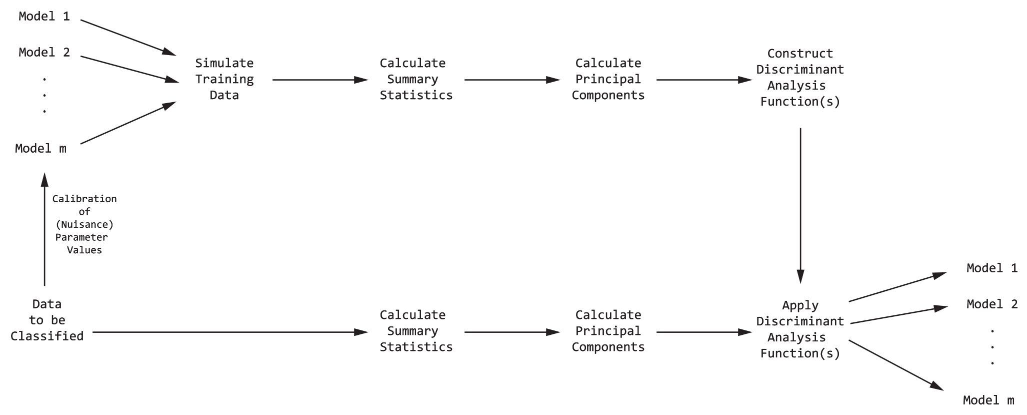
## 2.2 Installation from Source

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

```
    python setup.py install
```

# 3 Usage

## 3.1 Overview of Workflow


Model 1
Model 2
Model m
Simulate Training Data
Calculate Summary Statistics
Calculate Principal Components
Construct Discriminant Analysis Function(s)
Calibration of (Nuisance) Parameter Values
Data to be Classified
Calculate Summary Statistics
Calculate Principal Components
Apply Discriminant Analysis Function(s)
Model 1
Model 2
Model m

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:

```
{
  "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": {
    "lineage_birth_rate": {
      "definition_type": "lambda_definition",
      "definition": "lambda lineage: 0.01",
      "description": "fixed: 0.01"
    },
    "lineage_death_rate": {
      "definition_type": "lambda_definition",
      "definition": "lambda lineage: 0.0",
      "description": "fixed: 0.0"
    }
  },
  "anagenetic_range_evolution": {
    "lineage_area_gain_rate": {
      "definition_type": "lambda_definition",
      "definition": "lambda lineage: 0.01",
      "description": "fixed: 0.01"
    },
    "lineage_area_loss_rate": {
      "definition_type": "lambda_definition",
      "definition": "lambda lineage: 0.0",
      "description": "fixed: 0.0"
    }
  },
  "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:

**“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:

**"lineage_birth_rate":**
A *lineage-specific rate function definition* (see below) specifying the speciation rate.

**"lineage_death_rate":**
A *lineage-specific rate function definition* (see below) specifying the global extinction rate (i.e., the death rate in a birth-death branching process submodel, *not* the "extinction" or area loss rate of the DEC biogeographical submodel).

**"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:

**"lineage_area_gain_weight":**
A *lineage-specific rate function definition* (see below) specifying the rate at which a lineage loses an area in its range. When set to a fixed value, this corresponds to the area gain rate in the BayArea model or the *d* or "dispersal" parameter in the DEC model.

**"lineage_area_loss_rate":**
A *lineage-specific rate function definition* (see below) specifying the rate at which a lineage loses an area in its range. When set to a fixed value, this corresponds to the area loss rate in the BayArea model or the *e* or "extinction" (really, "extirpation") parameter in the DEC model.

**"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 Rate Function

A number of process rates (speciation, extinction, area gain, area loss) are specified in terms of a "lineage-specific rate 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 rate 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 Rate 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 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": 0.01,
}
```

The actual value of the "definition" entry would, of course, vary depending on the model. Typically, if not of analytical interest, this would be estimated based on the target data. Fixing the rate of a process to a value estimated from the target data is known as "calibrating" the model. More sophisticated approaches can be used to take into account uncertainty in the rate estimates. For example, sets of simulations can be run under different values of the process rate sampled from a prior.

#### 3.2.2.2 Trait State Mapping Lineage-Specific Rate Function

Here, you provide a list of rate 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 rate of 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],
}
```

#### 3.2.2.3 Lambda-Expression Rate Function

The rate 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 rate of the process for this particular lineage 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".

As another example, the following inspects the distribution vector of a lineage and returns a rate 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

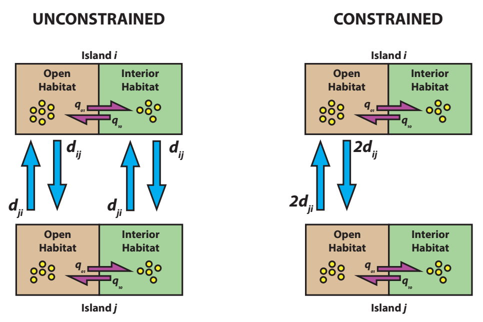
## 3.5 Profiling the Target Data

## 3.6 Encoding the Target Data

To calculate the summary statistics or profile the target data using `archipelago` the target phylogeny needs to be encoded with the geographical and trait information. This can

# 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 re-invade the interior habitats on those new islands by, again, moving into them in ecological space over evolutionary time.

## 4.2 The Target Data

## 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 largrange 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 unmodified. In the "constrained" class of model, we will use a trait-state mapping function which will return a rate of 0 if the lineage habitat trait is in "interior" state and *twice* the global dispersal rate estimated if the lineage trait is "disturbed" state. We use twice the rate so that the total dispersal flux across the system is maintained, given that, under an equal-rates traits model, we expect half the lineages to be in "interior" habitat state and consequently only half the lineages (the ones in the "disturbed" habitat state) to be actively 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 Calibrating the Training Data 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