# Biological shape as a weighted network

Peter D Smits Ben Frable

August 11, 2012

## 1 Brief Introduction

Here we envision biological shape, and the differences between shapes, as a network or graph model. A network model is the depiction of the relationships between many nodes. Each of these nodes may represent many different items or subjects of the same class. In, for example, a social network is where each node represents a single person. In this case, nodes represent a unique biological shape.

Between these nodes are links or edges. In the social network example, an edge would represent a "friendship" or some other social connection between two individuals. It is possible for edges to have weights, representing the degree of connectedness between two nodes. Unlike in a distance matrix, a higher value represents a stronger connectedness. Here, our edges represent the similarity of two shape configurations. How these edges are assigned and weighted, however, is a via an extended procedure.

In network theory, a community is a densly connected set of nodes that are more connected to each other than to other groups. In a given network, the number o groups may range between one large group and as many groups as there are nodes. The shape network made from the matrix of $\rho$ can then be analyzed using the various community detection algorithms in order to find if there are any subgroups of morphologies that are more densly connected. These subgroups represent highly similar morphologies, and different communities/groups represent large scale differences in morphology. Effectively, these community detection algorithms are defining different morphological categories directly from the morphological similarities as opposed to using prior knowledge or observation.

## 2 Methodology

The core of this entire process lies with the Riemannian shape distance, $\rho$, between two configurations. A shape configuration is the set of landmarks that represent a unique shape. There are a technically infinite number of unique shapes, and our sample in geometric morphometric studies only represents a subset of the biologically known or possible shapes.

Shapes, when properly scaled and fit via some form of Procrustes analysis, exist not in a Euclidean plane, but on a Riemannian manifold. Convenitnetly, a Riemannian manifold is metrizable and we can define some kind of metric to represent the "distance" between two shape configurations. This Riemannian shape distance, $\rho$ was defined by Kendall CITATION and varies between 0 and $\pi/2$. A value closer to 0 means that two shapes are closer in shape space, while a higher value indicates that two shapes are farther away in shape space. Conveniently here is a function for calculating $\rho$ in the `shapes` package CITATION called `riemdist()`.

The first step is the using generalized Procrustes analysis (GPA) to align all of our shapes and remove all non-shape elements. After this a pairwise distance matrix is calculated for all shape combinations using `riemdist()`. This produces a square $n \cdot n$ matrix, where $n$ is the number of unique shapes. This pairwise $\rho$ matrix represents how close every shape is to all other shapes. From here, we want to determine which shapes are closer to each other than we would expect by random. We require a Monte Carlo process to determine what "random" is.

How we define "by random" is very important here. Here, we are interested in a value of $\rho$ smaller than than we would expect half of the time by random. This means, that we want to find which values of $\rho$ are smaller than the median of a null distribution.

## 2.1 Monte Carlo process

The null distribution here is created from using two randomly selected shape configurations, moving the landmarks slightly to create two new, most likely valid, shape configuration. $\rho$ is then calculated for these two new shapes and stored. This process is repeated many times in order to approximate a distribution of possible values of $\rho$. After determining this null distribution, all values greater than the median of this null distribution are removed. The inverse of all remaining pairwise comparisons are then calculated, giving the future edge weights. The resulting matrix is an undirected, weighted adjacency matrix which can easily be converted into a network for use in the `igraph` package CITATION.

I've implemented this Monte Carlo process as a few functions in R. The core function is `jitter.landmark()`.

```
jitter.landmark <- function(land) {
    for (ii in seq(ncol(land))) {
        land[, ii] <- land[, ii] + runif(n = length(land[, ii]), min = -(min(abs(diff(land[,
            ii])))), max = min(abs(diff(land[, ii]))))
    }
    land
}
```

This functions are called `nsim` times (deafult 1000) by the `riem.distribution()` function. Using the original data set, `jitter.landmark()` is called on `nsim` two shape combinations. A lambda function is used to calculate the reimanian distance for all of the `nsim` combinations. Currently, the lambda function executed as part of a call to `mclapply()` which is a parallel function from the `parallel` core package. Currently, it will use the maximum number of cores avaliable and the seed is optimzed for reproducibility. Without this feature, the process

will run much slower. This function will not work on non-POSIX systems (i.e. Windows).

```
riem.distribution <- function(fit.land, nsim = 1000) {
    ran.land <- vector(mode = "list", length = nsim)
    for (ii in seq(nsim)) {
        s <- sample(fit.land$n, 2)
        sh1 <- jitter.landmark(fit.land$rotated[, , s[1]])
        sh2 <- jitter.landmark(fit.land$rotated[, , s[2]])
        ran.land[[ii]] <- abind(sh1, sh2, along = 3)
    }
    res <- mclapply(ran.land, function(x) {
        riemdis(x[, , 1], x[, , 2])
    }, mc.cores = detectCores(), mc.set.seed = T)
    res
}
```

The interface function `shape.simulation()` outputs a quantile value necessary for determining our "at random." The deafult quantile of 0.50 is the same as the median of a sample.

```
shape.simulate <- function(fit.land, nsim = 1000, probs = 0.5) {
    ri.di <- riem.distribution(fit.land, nsim = nsim)
    quantile(unlist(ri.di), probs = probs)
}
```

### 2.2 Community detection

There are currently eight different community detection algorithms implemented as part of the `igraph` package. Each of these algorithms try to optimze different communities based on various criteria. For example, the "walktrap" algorithm determines communties by which sections of a graph a random walk is most frequently. This algorithm, however, requires that there are no unconnected nodes, or nodes that have no edges. Currently, I have tested the "edge betweenness," "fast-greedy," and "leading eigenvector" algorithms. Each gives slightly different results, reflecting the method by which they determine couuminity presence and structure.

Implementation of these algorithms is illustrated below with gorilla and fish examples. Additionally, I've included heirarchical clustering results from the pariwise $\rho$ matrix for both of these examples.

## 3 Examples

### 3.1 Gorilla vertebrae

TODO

### 3.2 Fish example

The first step is to import the correct libraries and data files. The functions described above are part of my script file, while the `readtps.R` file is my fork of Ananda Mahto's original script GIST LINK. This fork improves integration

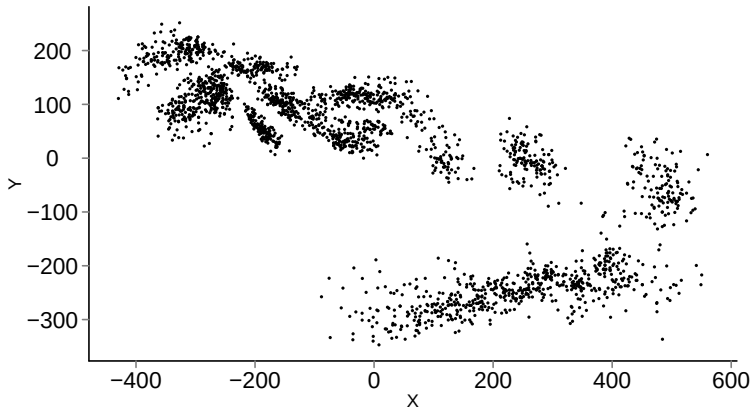


Figure 1: Landmark positions of 66 of the fish taxa. One specimen is excluded.

of .tps files and the `shapes` package. Additionally, becreporause this is a randomization process, for reproducibiliity it is necessary to manually set the seed value. Additionally, the RNG kind is changed to the L'Ecuyer algorithm CITATION which is the best for random number generationalgorithm implemented in R for parallel processes.

```
require(shapes)
require(igraph)
require(parallel)
require(abind)
source("landmark_montecarlo.R")
source("/home/peter/Documents/projects/3294254/readtps.R")
RNGkind("L'Ecuyer-CMRG")
set.seed(1)
ano <- read.tps("AnostomidaeFinal.TPS", shapes = TRUE)
cur <- read.tps("CurimatidaeF.TPS", shapes = TRUE)
fish <- abind(ano$landmark, cur$landmark)
```

Next, we use GPA to fit our entire sample of 67 fish. We also calculate the pairwise $\rho$ matrix. This is actually really dirty code that can probably be implemented as a single `outer` call, but I've yet to figure it out.

```
fish.fit <- procGPA(fish)
riem.comp <- matrix(0, ncol = fish.fit$n, nrow = fish.fit$n)
samp <- fish.fit$n
for (ii in seq(samp)) for (kk in seq(samp)) {
    riem.comp[ii, kk] <- riemdist(fish.fit$rotated[, , ii], fish.fit$rotated[,
        , kk])
}
```

From this fit, we should examine our data to better understand what is going on initially. The basic ways are looking at the fitted landmarks (Figure 1), the eigenscores from pricipal components analysis (Figure 2), and the thin-plate spine plot of an example fish to the mean shape (Figure 3.

Following this, the Monte Carlo process is run so that we can begin to analyze the structure of our shape relationships. This process is very long and takes

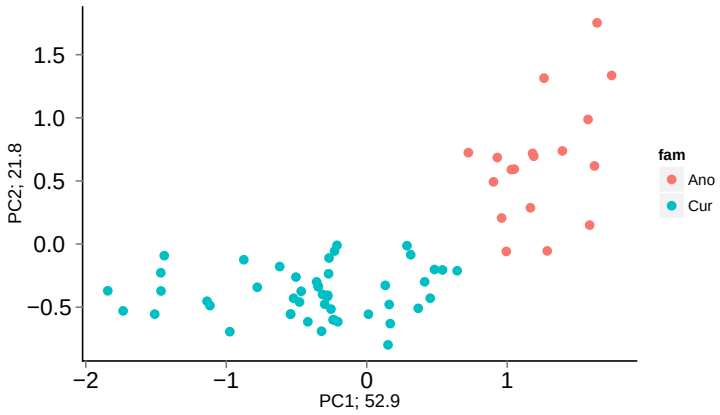


Figure 2: PC1 versus PC2 for fish. Families are labeled. Percent of variation explained is included in axes labels. One specimen (42) is excluded.

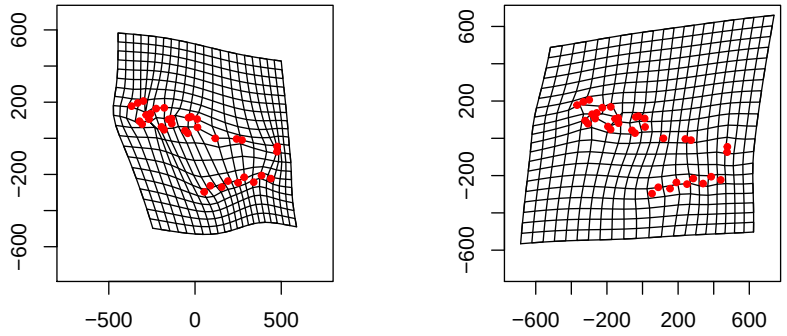


Figure 3: TPS grids for Ano (right) and Cur (left).

many hours. Those values which are greater than the median of the Monte Carlo distribution are discarted and the inverse of the remaining ones is calculated, producing a weighted adacency matrix.

```
cutoff <- shape.simulate(fish.fit, nsim = 10, probs = 0.5)
riem.comp[riem.comp > cutoff] <- 0
riem.comp[riem.comp != 0] <- 1/riem.comp[riem.comp != 0]
```

The weighted adacency matrix is then transformed into a network and the three community detection algorithms are then used to analyze the resulting graph.

```
data.graph <- graph.adjacency(riem.comp, mode = "undirected", weighted = TRUE)
data.fc <- fastgreedy.community(data.graph)
data.ebc <- edge.betweenness.community(data.graph)
data.lec <- leading.eigenvector.community(data.graph)
```

We can then examine the results of these community detection algorithms. The simplest manner is the just view the graphic output.

In general, there are two identified communities. These communities are not, however, identical to the two known families. There is a group of "Cur" fish that are most morphologically similar to "Ano" fish.

The next step after this is determining if this graph is actually interesting. This requires more reading.

# 4 Concerns and Future Development

1. I'm unsure how valid it is to define "by random" in the way that we did. I would need to actually talk to someone or read a lot to answer this one.
2. Need to make sure I'm not making invalid shape configurations
3. Unsure if I need to do `procGPA()` every time in the MC process.
4. Choice of community detection algorithm. This one just requires a lot of reading I think to figure out how exactly they differ.
5. Are the edge weights to high? Should they be rescaled
6. MC approach to understanding network. `statnet` packages. Is the network different from random?
7. Model based clustering possibilities. Can we classify without a network? Would use the `mclust` package CITATION, but there are awkward philosophical questions that come up with using that methodology.
8. Need to do some machine learning reading. This project is essentially unsupervised clustering, but doing it in a weird way.

For reproducibility information.

```
sessionInfo()

## R version 2.15.1 (2012-06-22)
## Platform: x86_64-pc-linux-gnu (64-bit)
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_AU.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_AU.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=C                 LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_AU.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] abind_1.4-0          igraph_0.6-2         shapes_1.1-5
## [4] MASS_7.3-20          rgl_0.92.892         scatterplot3d_0.3-33
## [7] formatR_0.6          knitr_0.7
##
## loaded via a namespace (and not attached):
## [1] digest_0.5.2   evaluate_0.4.2 plyr_1.7.1     stringr_0.6.1
## [5] tools_2.15.1
```