Biological Networks – Static Aspects

Preconditions

Make sure that the following software is installed on your `Linux` computer.

1. `gcc` to compile `R` add-on packages (e.g. `igraph`).
2. `latex` to compile the documentation for `R` add-on packages.
3. `emacs` or any other text-editor.
4. `R` a language / environment developed for statistical computing.
5. **The three data files (`Rintro.csv`, `Aminoacids.csv`, `YeasS.net`) used in this tutorial can be found online under the following URL**
   `http://www.tbi.univie.ac.at/~xtof/Leere/269020/`

The following directories and files are supposed to exist on your Linux computer.

```
$ test -d $HOME/local/bin
$ test -d $HOME/local/share/R
$ test -f $HOME/.Renviron

$ echo $PATH
/usr/local/bin:/usr/bin:$HOME/local/bin

$ cat  $HOME/.Renviron
R_LIBS=~/local/share/R
```

1 Basic facts on R

The first place to look for help, tutorials, online manuals or Wiki pages is the **R project homepage**

`http://www.r-project.org/`

Searching for R related content in Google can obviously make problems, therefor use the specialized search engin

`www.rseek.org/`

The place to look for **add-on packages** for R is CRAN (Comprehensive R Archive Network)

`http://cran.r-project.org/`

2 A first R session

Start R by typing R on the command line in the terminal

```
$ R
R version 3.4.3 (2017-11-30) -- "Kite-Eating Tree"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)
...
.
.
Type 'q()' to quit R.

>
```

resulting in a greeting and the R prompt, which is the `>` sign. Quit R by typing `q()` on the R prompt.

```
> q()
Save workspace image? [y/n/c]:
```

Answer `y` if you want so save all the variables in your workspace for a future session, `n` otherwise.

If you want to **interrupt a long running computation** and return to the R prompt without exiting R press `Ctrl-C`.

2.1 Getting Documentation and Help on the Web

You can **read the documentation** by typing

```
> help.start()
```

on the R prompt, which opens a browser window on the top-level table of contents of the documentation supplied with R. The following two links in the Reference section are especially usefull:

Packages Click here to see a list of all installed packages. Click on package names to see a list of its functions and datasets.

Search Engine & Keywords Click here to access a simple search engine allowing to search the local dcumentation for keywords or phrases.

Use the function `RSiteSearch()` to search the web for a *keyword* or *phrase*, which opens a browser window and directs it to the search engine on the R project website

```
> RSiteSearch("chemistry")
```

2.2 Getting Documentation or Help at the R prompt

You can **display information** on a particular *functionname* or *packagename* at the R prompt using (try e.g. function `mean` or package `base`)

```
> ?functionname
> help(functionname) or help(package="packagename")
> args(functionname)
> example(functionname)
```

or search the supplied documentation for a *pattern* (i.e. typically a *functionname* or *keyword*) using

```
> help.search("pattern")
```

2.3 Installing add-on R packages

Add-on R packages can either be installed from downloaded sources on the command line (1.), or alternatively directly from within R (2.)

1. Installation add-on from downloaded source

   ```
   $ R CMD INSTALL FOO-pkg.tar.gz -l ~/local/share/R
   ```

2. Install from within R

   ```
   > install.packages("FOO-pkg", lib="~/local/share/R")
   ```

Loading an R add-on package to use it's predefined functions and symbols

```
> library("FOO-pkg")
```

3 Brief R primer

3.1 Printing the Value of a Variable or Expression

Simply enter the *variable-name* or *expression* at the command prompt (R evaluates the variable or expression and then implicitely calls the function `print`)

```
> pi
 [1] 3.141593
> sqrt(2)
 [1] 1.414214
```

3.2 Setting Variables

Use the assignment operator (`<-`), there is no need to declare your variable first:

```
> x <- 3
> y <- 4
> x^2 + y^2 -> tmp
> tmp
 [1] 25
> z <- sqrt(tmp)
> print(z)
 [1] 5
```

3.3 Creating a Vector

Use the `c(...)` operator construct a vector from given values.

```
> v1 <- c(1,1,2,3,5,8,13,21)
> v1
 [1]  1  1  2  3  5  8 13 21
> v2 <- c(1,2,3)
> v3 <- c("A","B","C")
> v4 <- c(v2,v3)
> v4
 [1] "1" "2" "3" "A" "B" "C"
```

Use the `m:n` expression or the functions `seq()` or `rep()` to create a sequence (vector) of numbers.

```
> v1 <- 0:10
> v1
 [1]  0  1  2  3  4  5  6  7  8  9 10
> mean(v1)
 [1] 5
> v2 <- seq(from=0, to=10, by=2)
> v2
 [1]  0  2  4  6  8 10
> mean(v2)
 [1] 5
> v3 <- rep(1, times=10)
> v3
 [1] 1 1 1 1 1 1 1 1 1 1
> mean(v3)
```

3.4 Selecting Vector Elements

For extracting elements from a vector you have the following methods:

- Elements by position: use `[]` to select elements by index.

- Exclude elements: use negative indexes to exclude elements.
- Multiple elements: use a vector of indexes.
- Elements by condition: use a logical vector for selection.
- Elements by name: use names if elementsa are named.

```
> fib <- c(0,1,1,2,3,5,8,13,21,34)  # generate vector of 10 numbers
> fib[5]                            # select elemet 5
[1] 3
> fib[4:9]                          # select elements 4 through 9
[1] 2 3 5 8 13 21
> fib[fib %% 2 == 0]                # select even elements
[1] 0 2 8 34
> fib[-(1:3)]                       # ignore elements 1 to 3
[1] 2 3 5 8 13 21 34
```

3.5 Computing Basic Statistics

Use one of the following functions `mean(x)`, `median(x)`, `sd(x)`, `var(x)`, `cor(x,y)`, `cov(x,y)` assuming `x` and `y` are vectors (`summary` is a generic R-function producing nice result summeries).

```
> x <- c(0,1,1,2,3,5,8,13,21,34)
> mean(x)
[1] 8.8
> median(x)
[1] 4
> sd(x)
[1] 11.03328
> var(x)
[1] 121.7333
> y <- log(x+1)
> cor(x,y)
[1] 0.9068053
> cov(x,y)
[1] 11.49988
> x <- c(x, NA)
> x
[1]  0  1  1  2  3  5  8 13 21 34 NA
> var(x)
[1] NA
> var(x, na.rm=T)
[1] 121.7333
```

All these funcions are picky about values that are **not available** (`NA`). Even one `NA` value in the vector causes these functions to return `NA` or halt (`na.rm=T` tells R to ignore the `NA` values).

3.6 Linear Least-Square Regression

1. Generate an appropriate data-set

```
N  <- 1000
u  <- rnorm(N)
x1 <- rnorm(N)
x2 <- 1 + x1 + rnorm(N)
y  <- 1 + x1 + x2 + u
x  <- x1 + x2
df <- data.frame(x, y)
```

2. Next visualize and "least-square fit" the generated data-set

```
> plot(df)
> fit <- lm(y ~ x, data = df)
> summary(fit)
Call:
lm(formula = x ~ y, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-2.75758 -0.62407 -0.04432  0.67022  2.52499

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.66305    0.03676  -18.04   <2e-16 ***
y            0.83217    0.01190   69.91   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.9203 on 998 degrees of freedom
Multiple R-squared:  0.8304,    Adjusted R-squared:  0.8302
F-statistic:  4887 on 1 and 998 DF,  p-value: < 2.2e-16
```

3. Plot the regression line

```
plot(df$x,df$y)
abline(fit,col="red")
```

4. Finally apply your gained knowledge to the analyse / visualize the dataset `Rintro.csv`. Either fetch the file from it's web-location using `wget`

```
$ wget -v http://www.tbi.univie.ac.at/~xtof/Leere/269020/Rintro.csv
```

or load the data directly from the web-location within `R`

```
> url  <- "http://www.tbi.univie.ac.at/~xtof/Leere/269020"
> file <- paste(url, "Rintro.csv", sep="/")
> data <- read.table(file, header=T, sep=",")
```

4 Multivariate Data Analysis

4.1 Principle Component Analysis

1. Download the amino acids data set `Aminoacids.csv` from it's web-location using `wget`

```
$ wget -v http://www.tbi.univie.ac.at/~xtof/Leere/269020/Aminoacids.csv
```

2. Look at the content of file `Aminoacids.csv` using either a text editor (e.g. `emacs`) or the `less` command.

```
$ less Aminoacids.csv
or
$ emacs  Aminoacids.csv &
```

3. Start the program `R` and read in the amino acids data set. (If you have a local copy of the file you can pass the double quoted filename as first argument to the `read.table` function).

```
$ R
> url  <- "http://www.tbi.univie.ac.at/~xtof/Leere/269020"
> file <- paste(url, "Aminoacids.csv", sep="/")
> data <- read.table(file, header=T, sep=",")
> data
    AA   LCE   LCF   FET   POL   VOL   ASA
1  Ala  0.23  0.31 -0.55 -0.02  82.2 254.2
2  Arg -0.79 -1.01  2.00 -2.56 163.0 363.4
3  Asn -0.48 -0.60  0.51 -1.24 112.3 303.6
4  Asp -0.61 -0.77  1.20 -1.08 103.7 287.9
5  Cys  0.45  1.54 -1.40 -0.11  99.1 282.9
6  Gln -0.11 -0.22  0.29 -1.19 127.5 335.0
7  Glu -0.51 -0.64  0.76 -1.43 120.5 311.6
8  Gly  0.00  0.00  0.00  0.03  65.0 224.9
9  His  0.51  0.13 -0.25 -1.06 140.6 337.2
10 Ile  1.20  1.80 -2.10  0.04 131.7 322.6
11 Leu  1.28  1.70 -2.00  0.12 131.5 324.0
12 Lys -0.77 -0.99  0.78 -2.26 144.3 336.6
13 Met  0.90  1.23 -1.60 -0.33 132.3 336.3
14 Phe  1.56  1.79 -2.60 -0.05 155.8 366.1
15 Pro  0.38  0.49 -1.50 -0.31 106.7 288.5
16 Ser  0.00 -0.04  0.09 -0.40  88.5 266.7
17 Thr  0.17  0.26 -0.58 -0.53 105.3 283.9
18 Trp  1.85  2.25 -2.70 -0.31 185.9 401.8
19 Tyr  0.89  0.96 -1.70 -0.84 162.7 377.8
20 Val  0.71  1.22 -1.60 -0.13 115.6 295.1
```

4. Select a named vector e.g. `"POL"` from a data.frame object

```
> data[,"POL"]
 [1] -0.02 -2.56 -1.24 -1.08 -0.11 -1.19 -1.43  0.03 -1.06  0.04  0.12 -2.26
```

```
[13] -0.33 -0.05 -0.31 -0.40 -0.53 -0.31 -0.84 -0.13

> data$POL
 [1] -0.02 -2.56 -1.24 -1.08 -0.11 -1.19 -1.43  0.03 -1.06  0.04  0.12 -2.26
[13] -0.33 -0.05 -0.31 -0.40 -0.53 -0.31 -0.84 -0.13
```

5. Inspect the data set visually by plotting the six descriptors against each other (see Figure 1); look for correlations.

```
> plot(data[2:7])
```

6. Quantify the dependencies between the descriptors by calculating the correlation matrix (Note: there are different correlation measures "pearson", "spearman", "kendall").

```
> corrMx <- round(cor(data[,2:7], method="pearson"), 2)
> corrMx
      LCE   LCF   FET   POL   VOL   ASA
LCE  1.00  0.96 -0.96  0.72  0.39  0.40
LCF  0.96  1.00 -0.96  0.77  0.28  0.29
FET -0.96 -0.96  1.00 -0.78 -0.26 -0.27
POL  0.72  0.77 -0.78  1.00 -0.35 -0.33
VOL  0.39  0.28 -0.26 -0.35  1.00  0.99
ASA  0.40  0.29 -0.27 -0.33  0.99  1.00
```

7. Perform the principal component analysis.

```
> pca <- prcomp(data[,2:7], scale=T)
> pca
Standard deviations:
[1] 1.93173794 1.47441317 0.21022266 0.18759090 0.11117840 0.05243827

Rotation:
           PC1          PC2         PC3         PC4          PC5         PC6
LCE  0.5137329 -0.001441777  0.49921110  0.16989599 -0.439266747  0.51482505
LCF  0.5077828 -0.073021735 -0.24743132 -0.81646500  0.070889841  0.06294205
FET -0.5057126  0.084696444  0.67770662 -0.52677680 -0.006336087  0.01615821
POL  0.3726450 -0.464993711  0.44921191  0.12421352  0.344091745 -0.55614484
VOL  0.2017980  0.623271881  0.05778186 -0.03282580 -0.433821942 -0.61497315
ASA  0.2071309  0.618712592  0.15858648  0.10256666  0.703831374  0.20795031
```

8. Look at the distribution of variation among the principal components.

```
> summary(pca)
Importance of components:
                          PC1   PC2     PC3     PC4     PC5     PC6
Standard deviation     1.932 1.474 0.21022 0.18759 0.11118 0.05244
Proportion of Variance 0.622 0.362 0.00737 0.00587 0.00206 0.00046
Cumulative Proportion  0.622 0.984 0.99162 0.99748 0.99954 1.00000
```

9. Visualize the variances explained by the principal components as bar-plot.

```
> plot(pca)
```

10. Show the linear combination of descriptors for e.g. the first principal component.

```
> round(pca$rotation[,1], 3)
   LCE    LCF    FET    POL    VOL    ASA
 0.514  0.508 -0.506  0.373  0.202  0.207
```

11. Extract the principal component scores for e.g the first principal component.

```
> predict(pca)[,1]
 [1] -0.4341259 -2.9031169 -1.9124244 -2.3973034  0.8454390 -1.1220569
 [7] -2.0462123 -1.1769563 -0.1732949  2.2142318  2.2241869 -2.5247890
[13]  1.4338553  2.9601071  0.2986546 -1.0847787 -0.4400207  3.6611134
[19]  1.4836940  1.0937972
```

12. Visualize the (scaled) first two principal components in a bi-plot (1st data column are used as data point labels principal component.

```
> biplot(pca, xlabs=data[,1], col=c("black", "gray"))
```

5 Working with graphs

5.1 Basics of `igraph`

In the following section we will explore the structure of biological networks using the `R` add-on package `igraph`

1. Install and load the add-on package `igraph` from within `R`.

```
$ R
> install.packages("igraph", lib="~/local/share/R", repos="https://cran.wu.ac.at/")
> library("igraph")
```

2. Create and draw a graph.

```
> g <- graph(c(1, 2, 1, 3, 2, 3, 3, 5), n=5)
> plot(g)
```

3. Print the vertex and edge lists.

```
> V(g)
+ 5/5 vertices, from 40a4675:
[1] 1 2 3 4 5
> E(g)
+ 4/4 edges from 40a4675:
[1] 1->2 1->3 2->3 3->5
```

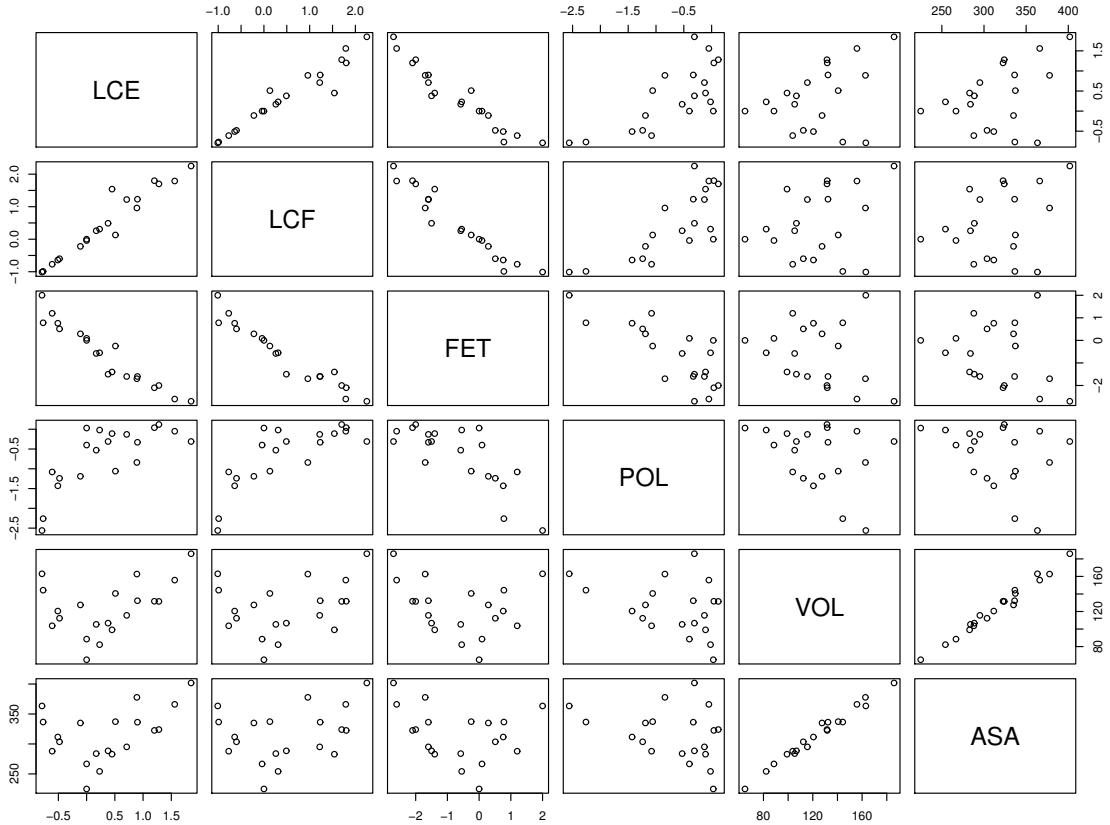

Figure 1: Scatter plots of the descriptors against each other. Note the high positive correlation between the two lipophilicity parameters (LCE, LCF) as well as between the volume (VOL) and the solvent-accessible surface area (ASA) parameter. FET (free energy of side chain tansfer from organic solvent to water) shows a highly negative correlation with the two lipophilicity parameters (LCE, LCF). POL characterizes polarity parameters.

4. Create a graph with 20 nodes and 15 random edges.

```
> g <- graph.empty() + vertices(letters[1:10], color="red")
> g <- g + vertices(letters[11:20], color="yellow")
> g <- g + edges(sample(V(g), 30, replace=TRUE), color="black")
> plot(g)
```

5.2 Basic network analysis

1. **Step-by-Step Example:**
   Generate a random network with suitable node and edge numbers.

```
> Y<-erdos.renyi.game(500,2000, "gnm",loops=T)
> Y<-as.undirected(Y,mode="arbitrary")
```

   A first sneak peek:

```
> plot(Y)
> vcount(Y)
> ecount(Y)
```

Now lets check some basic network statstics including degree distribution, clustering, modularity and average path length.

```
> plot(degree.distribution(Y),log="xy")
> max(degree(Y))
> average.path.length(Y)
> transitivity(Y)
> wtc<-cluster_walktrap(Y)
> modularity(Y,membership(wtc))
```

Finally, let's work a little on the layout - many possibilities are available in the igraph and other packages.

```
#circle layout
> plot(Y,layout=layout.circle,vertex.label=V(Y)$name,vertex.size=1)
#remove autoloops
> yy<-simplify(Y)
> yy<-delete.vertices(yy, V(yy)[degree(yy)==0])
> plot(yy,layout=layout.circle,vertex.label=V(yy)$name,vertex.size=1)
#reingold tilford
> plot.igraph(yy,vertex.size=3,vertex.label=NA,
layout=layout.reingold.tilford(yy, circular=T))
```

2. **Exercise 1: Explore the properties of the Yeast protein-protein network**
   Examine the protein-protein interaction network published by Bu et al (2003)[1].
   Load the yeast protein interaction network from file (*Pajek* is a common file format for large graphs).

```
> url  <- "http://www.tbi.univie.ac.at/~xtof/Leere/269020"
> file <- paste(url, "YeastS.net", sep="/")
> g <- read.graph(file, format="pajek")
```

   Count the number of vertices and edges. Remove regulatory autoloops. Calculate the vertex degrees (in , out, both) and plot the degree distribution (is it scale-free?). Find the highest connected nodes and identify their function. Determine the longest path (often called the diameter of the network) and the avarage path length between vertices (is it small-world?). Determine the clustering coefficient (also called transitivity) for each vertex and plot a histogram. What is the clustering coefficient of the entire network?

[1] Bu D et al (2003), Topological structure analysis of the protein-protein interaction network in budding yeast, *Nulc Acids Res* **31**(9):2443-2450 | doi:10.1093/nar/gkg340

3. **Exercise2: Everything links to everything - How to evolve natural network structure?**

   To get a better intuition on the dependencies of network properties, vertices and edges, we will evolve a network by hand starting from a small Erdos-Renyi graph (n=25, e=100). The goal is to strategically insert and delete edges and nodes such that one of the major features of adapted networks is obtained - power law in degree distribution.

Appendix

Exercise 1: Explore the properties of the Yeast protein-protein network

1. Count the number of vertices and edges.

```
> vcount(g)
[1] 2361
> ecount(g)
[1] 7182
> g<-simplify(g)
> ecount(g)
[1] 6646
```

2. Calculate the vertex degrees (in, out, both) and plot the degree distribution (is it scale-free?).

```
> d <- degree(g, mode="all")
> plot(d)
> ddist <- degree.distribution(g, mode="all")
> plot(ddist, log="xy", xlab="vertex degree", ylab="cummulative frequency")
> powerlaw <-power.law.fit(ddist)
> lines(1:50, (1:50)^((-powerlaw$alpha)))
> max(degree(g))
> V(g)$name[degree(g)==64]
[1] "YIL035C" "YPR110C"
https://www.yeastgenome.org/locus/S000001297
https://www.yeastgenome.org/locus/S000006314
```

3. Determine the longest path (often called the diameter of the network) and the avarage path length between vertices (is it small-world?).

```
> diameter(g)
[1] 11
> average.path.length(g)
[1] 4.376182
```

4. Determine the clustering coefficient (also called transitivity) for each vertex and plot a histogram. What is the clustering coefficient of the entire network?

```
> hist(transitivity(g, type="local"), breaks=100)
> transitivity(g)
[1] 0.1023149
```

Exercise 2: Everything links to everything - How to evolve natural network structure by hand?

1. Starting with a small random network and changing edges by hand following roughly a duplication-rewiring process.

```
> library(igraph)
> set.seed(123)
> n<-erdos.renyi.game(25,100,type="gnm")
> plot(n, layout=layout.circle)

#duplication
> n<-add.vertices(n,20)
> V(n)$name<-c(1:45)
> get.vertex.attribute(n)
#find candidates for wiring: node names - degree information
> sort(degree(n))

#visualize
> plot(n,layout=layout.circle,vertex.label=V(n)$name)

#check degree distribution
> plot(degree.distribution(n),log="xy")

#wiring
> n<-add.edges(n,c(45,18,44,18,43,18,42,21,41,21,40,13,39,9,38,9,37,3,36,25,35,
20,34,20,33,17,32,14,31,23,30,19,29,12,28,16,27,5,26,22))
> plot(degree.distribution(n),log="xy")

#find candidates for REwiring
> sort(degree(n))

#REwiring
#reconnect low degree nodes
> n<-add.edges(n,c(44,43,44,42,44,41,44,40,44,39,38,39,45,40,35,42,31,28,33,22))
> plot(n, layout=layout.circle,vertex.label=V(n)$name)
> plot(degree.distribution(n),log="xy")

#delete overrepresented by edge index
> E(n)
> n<-delete_edges(n,64)
> n<-add.edges(n,c(14,32))

#duplication-REwiring
> n<-add.vertices(n,6)
> V(n)$name<-c(1:51)
> n<-add.edges(n,c(46,39, 47, 45, 48,42,26,49,50,26,51,26,48,28,27,31,29,27))

#check powerlaw distribution
> powerlaw <-power.law.fit(degree.distribution(n, mode="all"))
> lines(1:15, (1:15)^((-powerlaw$alpha)+1))
```

```
#Moreover: some tools for removing edges of overrepresented nodes
#find candidates by degree
#V(n)$name[degree(n)==12]
#[1] 12 13
#in the edge table

##remove edges with name X
#n<-delete_edges(n,which(ends(n, E(n))[,1]==13))
#plot(degree.distribution(n),log="xy")
#remove edges by index
```

2. After this exercise, we realize that the emergence of similar properties in a broad range of evolved networks derive from underlying statistical process rules of evolution.