Lab2 732A51 Bioinformatics Group 9

Duc Duong, Martin Smelik, Raymond Sseguya

2018 M11 27

Question 1: DNA sequence acquisition and simulation

1.1 Simulate an artificial DNA sequence dataset

Firstly, the code block below will get the lizards DNA from GenBank, and save it as a fasta file. (code provided by the teacher)

```
## Gene bank accession numbers taken from http://www.jcsantosresearch.org/Class_2014_Spring_Comparative/
lizards_accession_numbers <- c("JF806202", "HM161150", "FJ356743", "JF806205",
                               "JQ073190", "GU457971", "FJ356741", "JF806207",
                               "JF806210", "AY662592", "AY662591", "FJ356748",
                               "JN112660", "AY662594", "JN112661", "HQ876437",
                               "HQ876434", "AY662590", "FJ356740", "JF806214",
                               "JQ073188", "FJ356749", "JQ073189", "JF806216",
                               "AY662598", "JN112653", "JF806204", "FJ356747",
                               "FJ356744", "HQ876440", "JN112651", "JF806215",
                               "JF806209")
lizards_sequences<-ape::read.GenBank(lizards_accession_numbers)
ape::write.dna(lizards_sequences, file ="lizard_seqs.fasta", format = "fasta", append =FALSE, nbcol = 6
print(lizards_sequences)
```

```
## 33 DNA sequences in binary format stored in a list.
##
## Mean sequence length: 1982.879
##    Shortest sequence: 931
##     Longest sequence: 2920
##
## Labels:
## JF806202
## HM161150
## FJ356743
## JF806205
## JQ073190
## GU457971
## ...
##
## Base composition:
##     a     c     g     t
## 0.312 0.205 0.231 0.252
## (Total: 65.44 kb)
```

We created a function called simulate_gene, which take the lizards sequences as the input and simulate an AI gene base on the original sequences. When calling the function, AI simulated gene is created automatically, and saved in a file called `AI_gene.fasta`. A message is returned to announce that the file is saved successfully.

```r
#1.1
set.seed(123)
clean <- function(template_gene){
  for (i in 1:length(template_gene)) {
    #Remove the " " that created when reading a file
    template_gene[[i]] <- template_gene[[i]][template_gene[[i]]!= " "]
  }
  return(template_gene)
}

# Read and clean
lizards_sequences <-  read.fasta("lizard_seqs.fasta")
lizards_sequences <- clean(lizards_sequences)

#Simulate AI gen function
simulate_gene <- function(template_gene)
{
  ai_gene <- list()
  gene_num <- length(template_gene)
  nucleotide <- c("a", "c", "g", "t")

  #Scan all gene and get some information
  for (i in 1:gene_num) {

    template_sequence <- template_gene[[i]]
    #get leng and base compotision of gene
    this_leng <- length(template_sequence)
    this_compotision = seqinr::count(template_sequence,1)/this_leng
    #print(this_compotision)

    #generate a new sequence base on sample function
    this_sequence <- sample(nucleotide, size=this_leng ,prob = as.vector(this_compotision), replace = TR
    #print(this_sequence)

    #add to list
    ai_gene[i] <- list(this_sequence)
  }

  #write to a file
  ape::write.dna(ai_gene, file ="AI_gene.fasta", format = "fasta", colsep ="")
  return("Created an AI gene and saved in file: AI_gene.fasta")
}

simulate_gene(lizards_sequences)
```

```
## [1] "Created an AI gene and saved in file: AI_gene.fasta"
```

Each sequence of the AI simulated gene has the same base composition with the original gene. For example, here is the base composition of the first sequence:

```r
ai_gene_1.1 <- read.fasta("AI_gene.fasta")
ai_gene_1.1 <- clean(ai_gene_1.1)
```

```r
print("Base composition of the first sequence of the original gene: ")
```

```
## [1] "Base composition of the first sequence of the original gene: "
```

```r
print(count(lizards_sequences[[1]],1)/length(lizards_sequences[[1]]))
```

```
##
##         a         c         g         t
## 0.2895792 0.2024048 0.2434870 0.2635271
```

```r
print("Base composition of the first sequence of the AI gene: ")
```

```
## [1] "Base composition of the first sequence of the AI gene: "
```

```r
print(count(ai_gene_1.1[[1]],1)/length(ai_gene_1.1[[1]]))
```

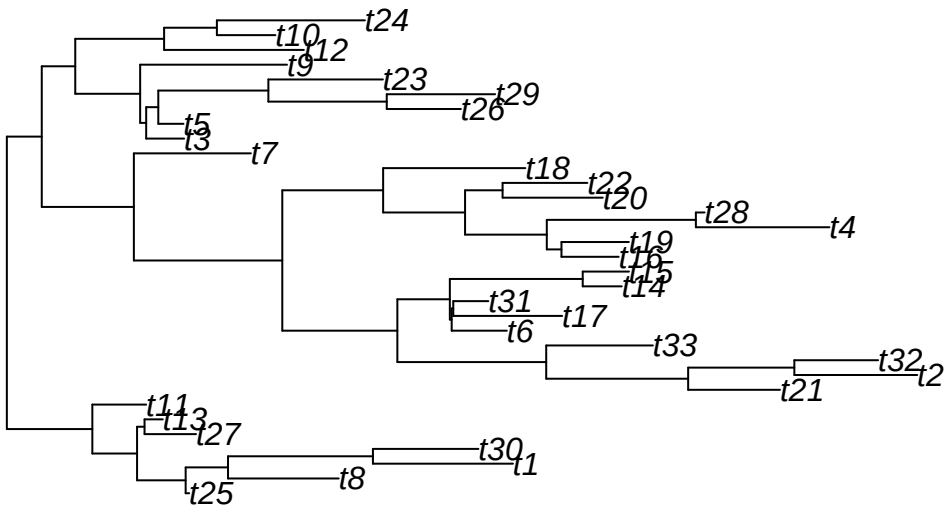
```
##
##         a         c         g         t
## 0.2905812 0.2024048 0.2364729 0.2705411
```

The composition is quite similar. There is a small different because the ai_gene only have “a” “g” “c” or “t”. While the original gene rearly has “y”, “m”, “r”... “y”" mean “c” or “t”, “m” means “a” or “c”, “r” means “a” or “g”.

1.2 Artificial DNA sequence dataset using phangorn::simSeq() function.

Here is the phylogenetic tree with 33 tips:

```r
#1.2
tree <- rtree(length(lizards_sequences))
plot(tree)
```



Then, we create a transition Q matrix base on random number around 0.25. Which quite similar with the true number of the real data. Here is our matrix rates

```
#the rates matrix
rates <- matrix(0, ncol = 4, nrow = 4)
rownames(rates) <- c("a", "c", "g", "t")
colnames(rates) <- c("a", "c", "g", "t")

#fill value to rates matrix
for (i in 1:4) {
  rate = runif(3, 0.22, 0.28)
  for (j in 1:4) {
    if (j==4)
    {
      rates[i,j]= 1- sum(rate)
    }else rates[i,j] = rate[j]
  }
}

#print the rates
rates
```

```
##           a         c         g         t
## a 0.2673809 0.2768651 0.2359109 0.2198432
## c 0.2542675 0.2610672 0.2559322 0.2287331
## g 0.2271230 0.2446746 0.2645644 0.2636380
```

```
## t 0.2320356 0.2207405 0.2357135 0.3115104
```

Finally, we create the second AI simulated gene base on the phangorn::simSeq() function. We choose the length of all sequences equals 1000, Which more or less like the original sequence.
The second AI simulated gene is saved as a fasta file with the name AI_gene2.fasta

```
#create the ai_gene 2
ai_gene_1.2 <- phangorn::simSeq(tree, l = 1000, Q=rates , type = "DNA")

#rename
for (i in 1:length(ai_gene_1.2)){
  ai_gene_1.2[[i]][ai_gene_1.2[[i]] == 1] = "a"
  ai_gene_1.2[[i]][ai_gene_1.2[[i]] == 2] = "c"
  ai_gene_1.2[[i]][ai_gene_1.2[[i]] == 3] = "g"
  ai_gene_1.2[[i]][ai_gene_1.2[[i]] == 4] = "t"
}

ape::write.dna(ai_gene_1.2, file ="AI_gene2.fasta", format = "fasta", colsep ="")
```

Question2: Sequence analysis

2.1: Report some basic statistics

Here is some basic statistics as the requirements for the frist sequence of each gene:

```
#2.1
ai_gene_1.2 <- read.fasta("AI_gene2.fasta")
ai_gene_1.2 <- clean(ai_gene_1.2)

#for (i in 1:length(lizards_sequences)) {
for (i in 1:1) {
  cat(paste("For the sequences number: ", i , "\n"))
  print("The composition of lizards dataset:")
  print(round(count(lizards_sequences[[i]],2)/length(lizards_sequences[[i]]), 4))

  print("The composition of AI_gene1 dataset:")
  print(round(count(ai_gene_1.1[[i]],2)/length(ai_gene_1.1[[i]]), 4))

  print("The composition of Ai_gene2 dataset:")
  print(round(count(ai_gene_1.2[[i]],2)/length(ai_gene_1.2[[i]]), 4))

  cat("\n")
}
```

```
## For the sequences number:  1
## [1] "The composition of lizards dataset:"
##
##     aa     ac     ag     at     ca     cc     cg     ct     ga     gc
## 0.0862 0.0481 0.0782 0.0772 0.0731 0.0501 0.0130 0.0661 0.0962 0.0501
##     gg     gt     ta     tc     tg     tt
```

```
## 0.0471 0.0491 0.0341 0.0531 0.1052 0.0701
## [1] "The composition of AI_gene1 dataset:"
##
##     aa     ac     ag     at     ca     cc     cg     ct     ga     gc
## 0.0902 0.0611 0.0611 0.0782 0.0561 0.0341 0.0581 0.0541 0.0711 0.0491
##     gg     gt     ta     tc     tg     tt
## 0.0501 0.0661 0.0721 0.0581 0.0671 0.0721
## [1] "The composition of Ai_gene2 dataset:"
##
##    aa    ac    ag    at    ca    cc    cg    ct    ga    gc    gg    gt
## 0.066 0.070 0.061 0.047 0.064 0.058 0.063 0.063 0.053 0.054 0.067 0.077
##    ta    tc    tg    tt
## 0.060 0.066 0.061 0.069
```

We can see that the composition for two nucleotide of AI simulated genes are not similar with the original sequence. It's understandable because we just create AI sequence base on the composition of each nucleotide.

Even additionally here below:

```
ape::base.freq(as.DNAbin(lizards_sequences), freq = FALSE, all = FALSE)
```

```
##         a         c         g         t
## 0.3121454 0.2052325 0.2307222 0.2518999
```

```
GC.content(as.DNAbin(lizards_sequences))
```

```
## [1] 0.4359547
```

```
ape::base.freq(as.DNAbin(ai_gene_1.1), freq = FALSE, all = FALSE)
```

```
##         a         c         g         t
## 0.3145411 0.2034997 0.2284404 0.2535188
```

```
GC.content(as.DNAbin(ai_gene_1.1))
```

```
## [1] 0.4319401
```

```
print(count(ai_gene_1.2[[1]],1)/length(ai_gene_1.1[[1]]))
```

```
##
##        a        c        g        t
## 0.244489 0.248497 0.252505 0.256513
```

We can see that the composition for the sencod AI gene is not similiar with the first ones and the original sequence. Because the second ones is created base on the Q matrix.

We also observed the stop codons both in the original and the simulated sequences. When using the **EMBOSS Transeq** online tool, the stop codons were marked by stars. Below in a snap shot of part of the original sequence for the **FJ356743 lizard**.

```
>FJ356743_1
QSKIT*EATF**QPSGYQQ*RKSGSFSGKKQ*GKDGDCGCKVTLTL*NRHRVEQKYSENR
*GCLSYEPNRG*NTPGKPAASLPHLWRLISN*PL*EKPPSSWTS***DSGPSQKEREKGN
ILARSPWQSF*D*CERGH*HNPSNKVLSPVLDCGSKKIKQFPM*TIFSKERPSGMASPFL
KL*CLWHFPPWSEEKEASP*STAEQKVEDGGWMY*KNKTDKETKTSQPKEFDEKNCQLQE
DSP*YQDPCSRLSRRLCKVNLLPDL*AHPS*PSRNDMQALILQNLHP*MPQSNGKLLPSL
SLSLLSY*SGEPCEILPEHPEQSTCEMSSDRLSGGGLLGKIL*PSFQTQRGRRQRGVCVR
KQRWPTKTTLTFTDTESSKAPLKRTQATSKSFC*ERRRGRCQVCVSNFVPTGSASQK*TQ
TS**VGSYNAREGVRPSSSCLLGNQSQYFSQL*PVP*NVQDCKSYNWETDIPATPCPPNC
*KVPLARLPSV*VETTLEKCLQ*HRSRHY*WAFRHTTFG**LPSRHNCKEISI*CRFGFC
LDGYGRRHPRRPEKSGPG*LSQGPFHCGD*RVL*WNGRC**ETWLWPSCP*ESSSILFHT
HEHLCHSWQCKQKDF*RK*TQFRTVL*TFVPYAG**IRP*DTHSHPESSCGRKRGHERQC
TDT*YGWNPKNVQIHI*RHWI**KACP*SRGP*SFRLYLHLHAV*CNTPRGLTELDPSLH
HKESCGKLRKVRGVEVQPLS*DC**TA*QSEGGFCKAFY*DCAFYRCLAL*HWQCS*ILQ
DISV*DR*SLQNPQCIKRREKEMAVSS*QTPQKEDELEACHQNEWEFCQKAHDQGDCGSS
L*INKE*GETRSPQRTRGPLP*DETSVAVFMSHQGMPRTSMPV*F*FSTFCGAVGYKIPL
QI*GQDYQLLSQNPRSCSRNY*KGWIYWCLGK*GK*VWEQTFQTLSKNECQTVKML*NGR
```

We observed **1269 stop codons** in the original sequence. Using set.seed(123), we observed **1332 stop codons** in the first AI simulated sequence and very surprisingly only **491 stop codons** in the second AI simulated sequence. Likewise, the amino acid composition changed with each simulation due to the randomisation.

Again, still using set.seed(123), when we reversed and complemented the sequences, plus using ORF. The smallest number of stop codon is recored as follow: **897 stop codons** for the original sequence, **1107 stop codons** for the first simulated sequence and **527 stop codons** for the second simulated sequence.

2.2: Markov chain

We decided to use the `markovchain` library to fit the markovchanin model. Here is the resutl:

```
clean2 <- function(template_gene){
  nucleotide <- c("a", "c", "g", "t")
  for (i in 1:length(template_gene)) {
    #Remove the " " that created when reading a file
    template_gene[[i]] <- template_gene[[i]][template_gene[[i]]!= " "]
    #Remove the character that not normal nucleotide (a, c, t, g)
    template_gene[[i]] <- template_gene[[i]][grepl(paste0(nucleotide, collapse = "|"), template_gene[[i]
  }
  return(template_gene)
}

lizards_sequences <- clean2(lizards_sequences)
library(markovchain)
markovchainFit(lizards_sequences)
```

```
## $estimate
## MLE Fit
##  A  4 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, t
```

```
##  The transition matrix  (by rows)  is defined as follows:
##           a         c          g         t
## a 0.3379074 0.1731438 0.27512864 0.2138201
## c 0.3794646 0.2477071 0.05018269 0.3226456
## g 0.3938349 0.2031820 0.19330461 0.2096785
## t 0.1509262 0.2116611 0.35718190 0.2802308
##
##
## $standardError
##             a           c           g           t
## a 0.004069401 0.002912965 0.003671974 0.003237099
## c 0.005319307 0.004297725 0.001934400 0.004904924
## g 0.005109571 0.003670033 0.003579715 0.003728244
## t 0.003027621 0.003585417 0.004657618 0.004125504
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c          g         t
## a 0.3312138 0.1683524 0.26908879 0.2084956
## c 0.3707151 0.2406380 0.04700088 0.3145777
## g 0.3854304 0.1971453 0.18741650 0.2035461
## t 0.1459462 0.2057636 0.34952080 0.2734449
##
## $upperEndpointMatrix
##           a         c          g         t
## a 0.3446009 0.1779352 0.28116850 0.2191447
## c 0.3882141 0.2547762 0.05336449 0.3307135
## g 0.4022394 0.2092186 0.19919271 0.2158109
## t 0.1559062 0.2175586 0.36484300 0.2870166
```

```
markovchainFit(ai_gene_1.1)
```

```
## $estimate
## MLE Fit
##  A  4 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, t
##  The transition matrix  (by rows)  is defined as follows:
##           a         c         g         t
## a 0.3146302 0.2049886 0.2274517 0.2529295
## c 0.3112981 0.2030499 0.2291917 0.2564603
## g 0.3197511 0.2016593 0.2266827 0.2519069
## t 0.3123605 0.2039573 0.2305604 0.2531218
##
##
## $standardError
##             a           c           g           t
## a 0.003911243 0.003157035 0.003325517 0.003506826
## c 0.004835780 0.003905528 0.004149328 0.004389231
## g 0.004625339 0.003673218 0.003894455 0.004105419
## t 0.004340853 0.003507653 0.003729404 0.003907615
##
## $confidenceLevel
```

```
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c         g         t
## a 0.3081968 0.1997957 0.2219818 0.2471612
## c 0.3033439 0.1966259 0.2223667 0.2492407
## g 0.3121431 0.1956174 0.2202769 0.2451541
## t 0.3052204 0.1981877 0.2244261 0.2466943
##
## $upperEndpointMatrix
##           a         c         g         t
## a 0.3210637 0.2101814 0.2329217 0.2586977
## c 0.3192522 0.2094739 0.2360167 0.2636800
## g 0.3273591 0.2077012 0.2330885 0.2586597
## t 0.3195006 0.2097269 0.2366947 0.2595493
```

```
markovchainFit(ai_gene_1.2)
```

```
## $estimate
## MLE Fit
##  A  4 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, t
##  The transition matrix  (by rows)  is defined as follows:
##           a         c         g         t
## a 0.2455564 0.2495739 0.2527392 0.2521305
## c 0.2495729 0.2473761 0.2454235 0.2576275
## g 0.2536724 0.2416986 0.2405876 0.2640415
## t 0.2489950 0.2545519 0.2435564 0.2528967
##
##
## $standardError
##             a           c           g           t
## a 0.005467619 0.005512165 0.005547010 0.005540326
## c 0.005518877 0.005494535 0.005472806 0.005607227
## g 0.005595866 0.005462201 0.005449633 0.005709088
## t 0.005425772 0.005485981 0.005366189 0.005468116
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c         g         t
## a 0.2365629 0.2405072 0.2436152 0.2430175
## c 0.2404951 0.2383384 0.2364215 0.2484045
## g 0.2444680 0.2327140 0.2316237 0.2546509
## t 0.2400704 0.2455283 0.2347298 0.2439024
##
## $upperEndpointMatrix
##           a         c         g         t
## a 0.2545498 0.2586406 0.2618632 0.2612435
## c 0.2586506 0.2564138 0.2544254 0.2668506
## g 0.2628768 0.2506831 0.2495514 0.2734321
## t 0.2579196 0.2635755 0.2523830 0.2618909
```

Markov chain order: We use the `fitHigherOrder` funcion. Based on the below result (for the first sequences), we concluded that all genes have a high order. For the lizard sequence, it make sense because we have 3 nucleotide for each protein code. So we should have at least 3rd order. For the AI genes, the high order could be by chance of probability

```
#2.2
#The fitHigherOrder function work only with a list, not list of list
fitHigherOrder(lizards_sequences[[1]])
```

```
## $lambda
## [1] 0.5 0.5
##
## $Q
## $Q[[1]]
##           a         c         g         t
## a 0.2975779 0.36138614 0.3950617 0.1297710
## c 0.1660900 0.24752475 0.2098765 0.2022901
## g 0.2698962 0.06435644 0.1934156 0.4007634
## t 0.2664360 0.32673267 0.2016461 0.2671756
##
## $Q[[2]]
##           a         c         g         t
## a 0.3425606 0.2089552 0.2921811 0.2938931
## c 0.1591696 0.1940299 0.2263374 0.2328244
## g 0.2733564 0.2587065 0.2222222 0.2213740
## t 0.2249135 0.3383085 0.2592593 0.2519084
##
##
## $X
##         a         c         g         t
## 0.2898696 0.2026078 0.2437312 0.2637914
```

```
fitHigherOrder(ai_gene_1.1[[1]])
```

```
## $lambda
## [1] 0.5 0.5
##
## $Q
## $Q[[1]]
##           a         c         g         t
## a 0.3103448 0.2772277 0.3008475 0.2676580
## c 0.2103448 0.1683168 0.2076271 0.2156134
## g 0.2103448 0.2871287 0.2118644 0.2490706
## t 0.2689655 0.2673267 0.2796610 0.2676580
##
## $Q[[2]]
##           a         c         g         t
## a 0.2793103 0.3118812 0.2297872 0.3382900
## c 0.2172414 0.1584158 0.2085106 0.2156134
## g 0.2310345 0.2277228 0.2340426 0.2490706
## t 0.2724138 0.3019802 0.3276596 0.1970260
##
##
```

```
## $X
##         a         c         g         t
## 0.2905812 0.2024048 0.2364729 0.2705411
```

```
fitHigherOrder(ai_gene_1.2[[1]])
```

```
## $lambda
## [1] 0.5 0.5
##
## $Q
## $Q[[1]]
##           a         c         g         t
## a 0.2704918 0.2580645 0.2111554 0.2343750
## c 0.2868852 0.2338710 0.2151394 0.2578125
## g 0.2500000 0.2540323 0.2669323 0.2382812
## t 0.1926230 0.2540323 0.3067729 0.2695312
##
## $Q[[2]]
##           a         c         g         t
## a 0.2172131 0.2500000 0.2549801 0.2470588
## c 0.2581967 0.2419355 0.2310757 0.2627451
## g 0.2418033 0.2419355 0.2908367 0.2352941
## t 0.2827869 0.2661290 0.2231076 0.2549020
##
##
## $X
##     a     c     g     t
## 0.244 0.248 0.252 0.256
```

2.3: Align the sequences

We use the `msa` library to align the sequences, then calculate the distantce and draw some heatmaps as below:

```
#2.3
library(msa)
#Original
real_align <- msaClustalW("lizard_seqs.fasta",type="dna")
```

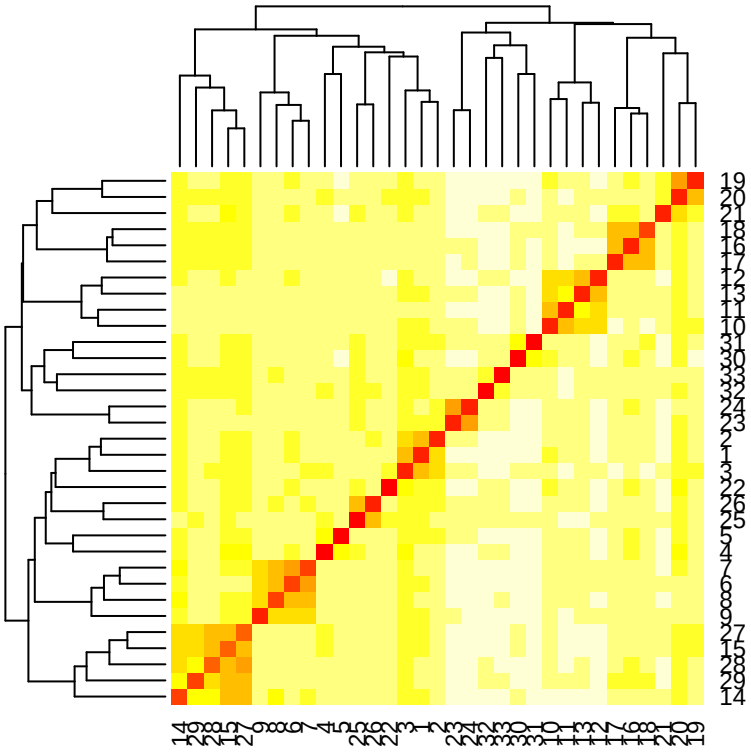
```
## use default substitution matrix
```

```
real_alignseq<- msaConvert(real_align, type="seqinr::alignment")
dist_real <- as.matrix(dist.alignment(real_alignseq, "identity"))

#Ai 1
ai1.1_align <- msaClustalW("AI_gene.fasta",type="dna")
```

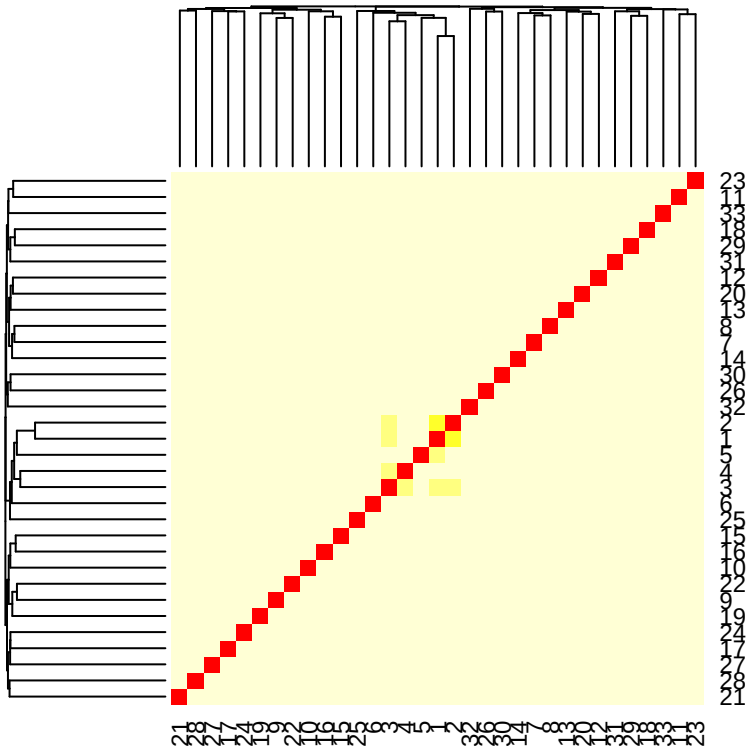
```
## use default substitution matrix
```

```
ai1.1_alignseq <- msaConvert(ai1.1_align, type="seqinr::alignment")
dist_a1.1 <- as.matrix(dist.alignment(ai1.1_alignseq, "identity"))

#AI 2
ai1.2_align <- msaClustalW("AI_gene2.fasta",type="dna")


## use default substitution matrix
```

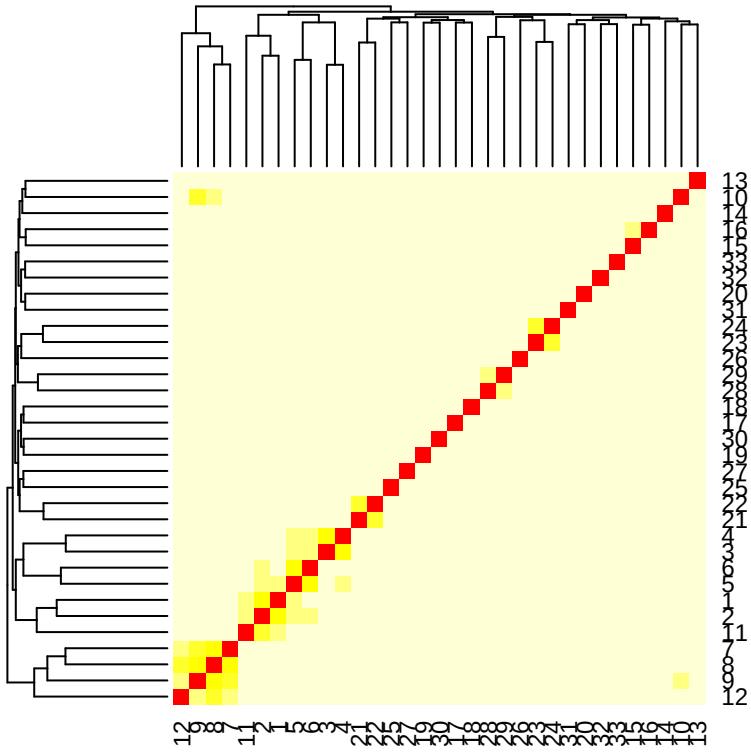
```
ai1.2_alignseq<- msaConvert(ai1.2_align, type="seqinr::alignment")
dist_a1.2 <- as.matrix(dist.alignment(ai1.2_alignseq, "identity"))

heatmap(dist_real)
```



```
heatmap(dist_a1.1)
```



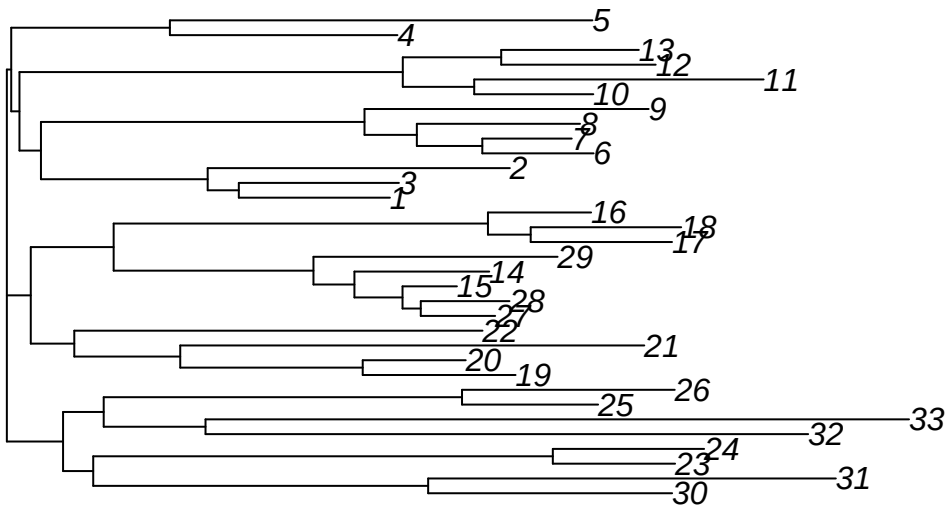

```
heatmap(dist_a1.2)
```



As we can see that the values in heatmap of AI simulated sequences are quite low. It means that the AI simulated gene has low (or no) connections with each other (because it randomly created) while the original gene is highly connected.

Question 3: Phylogeny reconstruction

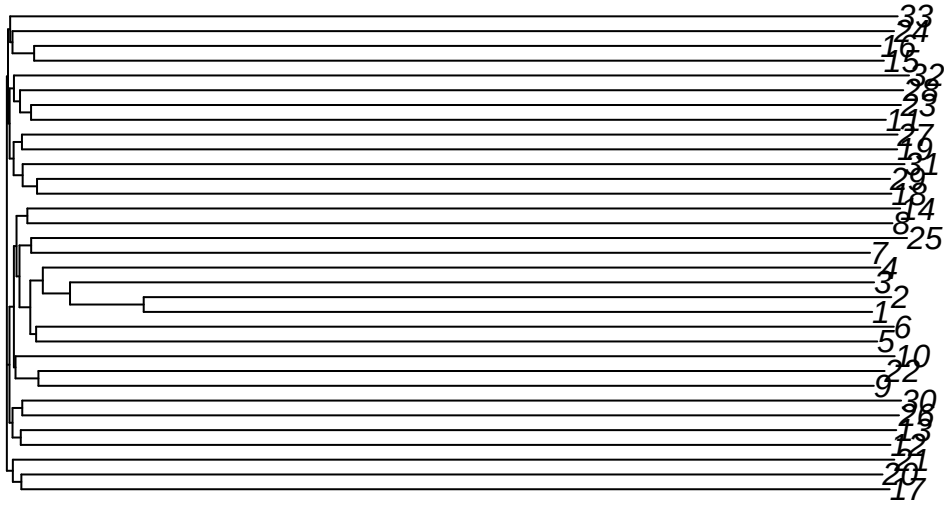
```
## with help from Roshni

dna <- as.DNAbin(real_alignseq)
D1 <- dist.dna(dna, model="TN93")
F = function(x) njs(x)
D1[is.nan(D1)] <- 10
tree1<-F(D1)
v1 <- boot.phylo(phy =  tree1, x = dna,
                 FUN = F, quiet = T,
                 1000)
plot(tree1)
```




```
v1
```

```
##  [1] 1000    0    0    1    0    3    0    0    0    0  100    2   53   43
## [15]   71    0    0    1   49   33    4    5   39    0   76   36    2   34
## [29]   29   38   36
```

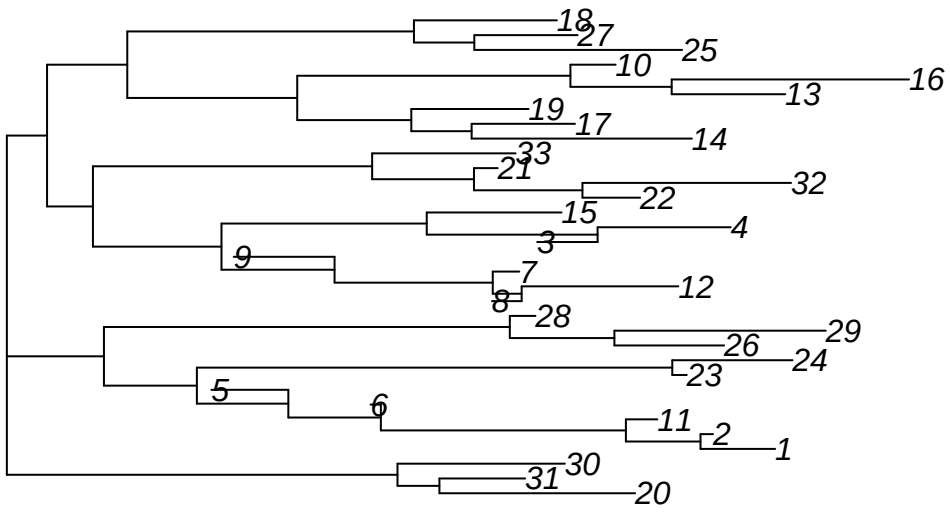
```
dna <- as.DNAbin(ai1.1_alignseq)
#D2 <- dist.dna(dna, model="TN93")
D2 <- dist.alignment(ai1.1_alignseq, "identity")
D2[is.nan(D2)] <- 10
tree2<-F(D2)
v2 <- boot.phylo(phy =  tree2, x = dna,
                 FUN = F, quiet = T,
                 1000)
plot(tree2)
```




```
v2
```

```
##  [1] 1000    0    0    0    0    1    0    0    0    0    6    2    0    0
## [15]   22    1    0   12   22   18   17    0   10   31   13   11    8   14
## [29]    2   12   45
```

```
dna <- as.DNAbin(ai1.2_alignseq)
D3 <- dist.dna(dna, model="TN93")
D3[is.nan(D3)] <- 10
tree3<-F(D3)
v3 <- boot.phylo(phy =  tree3, x = dna,
                 FUN = F, quiet = T,
                 1000)
plot(tree3)
```




```
v3
```

```
##  [1] 1000    0    0    0    0    0    0    0    0    2    0    0    2   15
## [15]    1   17    0   30    0    0    2   24    0    3    2   12   23   21
## [29]   25   26   27
```

At the pictures we can see 3 quite different trees. First one, created by lizard sequences is showing quite deep branches with maximum depth of 7. This shows that the lizards are really connected to each other from the point of view of the evolutionary model. This tree has also the best result in bootstraping therefore we can say that in some way this tree is the most stable. It is easy to see in the second tree that the sequences are random as there is almost no connection between the sequences(Just joining two letters together makes 4*4=16 possibilities and with just 33 sequences it is difficult to find similarities). Bootstraping shows that the tree changes quite much with small changes in sequences.
The last tree is created by a sequences that were created according to the tree and therefore we can see that this tree has quite similar depth in each branch. The bootstraping showed that with minor changes in the sequences the tree can still look sometimes quite similiar.
Comparing these trees with the one simulated in 1.2 we can see that the one in 1.2 is much deeper in most of the branches.

```
treedist(tree1, tree2, check.labels = TRUE)
```

```
##      symmetric.difference   branch.score.difference
##                 56.000000                  2.278781
##           path.difference quadratic.path.difference
##                 73.539105                 17.811108
```

```r
treedist(tree1, tree3, check.labels = TRUE)
```

```
##      symmetric.difference   branch.score.difference
##                 58.000000                  8.587466
##           path.difference quadratic.path.difference
##                 76.092050                172.198367
```

```r
treedist(tree2, tree3, check.labels = TRUE)
```

```
##      symmetric.difference   branch.score.difference
##                 58.000000                  7.963635
##           path.difference quadratic.path.difference
##                 79.962491                155.268303
```

We can see different distance measures used between the trees. It is interesting to see that the minimum and maximum combinations are different for each measure. “‘