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.

```
#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 = T
    #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:

```
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]]))
```

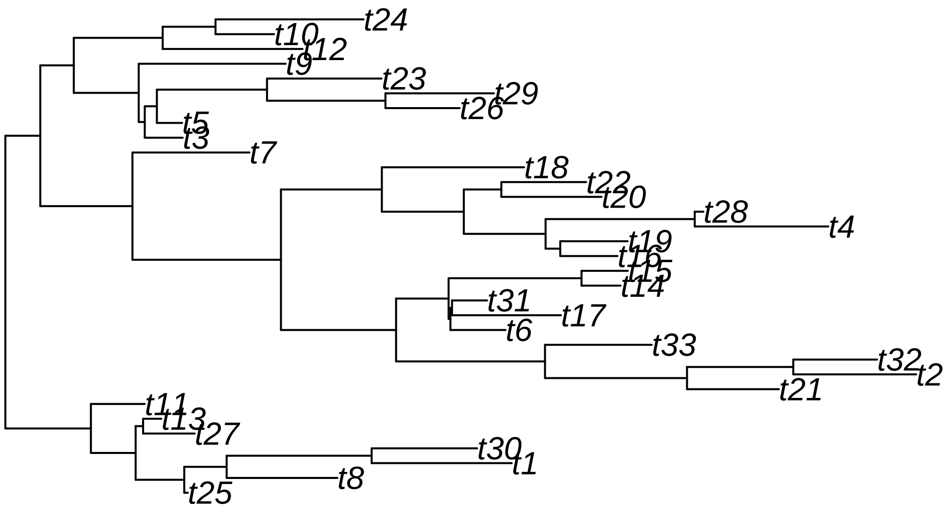
```
##
##         a         c         g         t
## 0.2905812 0.2024048 0.2364729 0.2705411
```

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"...

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 genes are not similar with the original sequence. It's understandable

2.2: Markov chain

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

```r
library(markovchain)
markovchainFit(lizards_sequences)
```

```
## $estimate
## MLE Fit
##  A  8 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, m, r, s, t, y
##  The transition matrix  (by rows)  is defined as follows:
##           a         c          g            m            r            s
## a 0.3377604 0.1730948 0.27493261 4.900760e-05 0.0002450380 0.000000e+00
## c 0.3793901 0.2477071 0.05010812 0.000000e+00 0.0003728283 0.000000e+00
## g 0.3934372 0.2029168 0.19323832 6.629102e-05 0.0003314551 0.000000e+00
## m 0.0000000 0.0000000 0.66666667 0.000000e+00 0.0000000000 0.000000e+00
## r 0.4117647 0.1764706 0.11764706 0.000000e+00 0.0000000000 0.000000e+00
## s 0.0000000 1.0000000 0.00000000 0.000000e+00 0.0000000000 0.000000e+00
## t 0.1508047 0.2115396 0.35718190 6.073489e-05 0.0001214698 6.073489e-05
## y 0.3333333 0.2000000 0.13333333 0.000000e+00 0.0000000000 0.000000e+00
##           t            y
## a 0.2136731 0.0002450380
## c 0.3222728 0.0001491313
## g 0.2096122 0.0003977461
## m 0.3333333 0.0000000000
## r 0.2941176 0.0000000000
## s 0.0000000 0.0000000000
## t 0.2801093 0.0001214698
## y 0.3333333 0.0000000000
##
##
## $standardError
##             a           c           g            m            r
## a 0.004068516 0.002912552 0.003670666 4.900760e-05 1.095843e-04
## c 0.005318784 0.004297725 0.001932963 0.000000e+00 1.667339e-04
## g 0.005106990 0.003667637 0.003579101 6.629102e-05 1.482312e-04
## m 0.000000000 0.000000000 0.471404521 0.000000e+00 0.000000e+00
## r 0.155632430 0.101885342 0.083189033 0.000000e+00 0.000000e+00
## s 0.000000000 1.000000000 0.000000000 0.000000e+00 0.000000e+00
## t 0.003026402 0.003584388 0.004657618 6.073489e-05 8.589211e-05
## y 0.149071198 0.115470054 0.094280904 0.000000e+00 0.000000e+00
##              s           t            y
## a 0.000000e+00 0.003235986 1.095843e-04
## c 0.000000e+00 0.004902089 1.054518e-04
## g 0.000000e+00 0.003727654 1.623792e-04
## m 0.000000e+00 0.333333333 0.000000e+00
## r 0.000000e+00 0.131533410 0.000000e+00
## s 0.000000e+00 0.000000000 0.000000e+00
## t 6.073489e-05 0.004124610 8.589211e-05
## y 0.000000e+00 0.149071198 0.000000e+00
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##            a           c          g m            r s          t
## a 0.33106824 0.168304107 0.26889491 0 6.478782e-05 0 0.20835040
## c 0.37064143 0.240637977 0.04692868 0 9.857546e-05 0 0.31420954
## g 0.38503694 0.196884079 0.18735122 0 8.763643e-05 0 0.20348075
## m 0.00000000 0.000000000 0.00000000 0 0.000000e+00 0 0.00000000
## r 0.15577214 0.008884115 0.00000000 0 0.000000e+00 0 0.07776444
## s 0.00000000 0.000000000 0.00000000 0 0.000000e+00 0 0.00000000
## t 0.14582675 0.205643836 0.34952080 0 0.000000e+00 0 0.27332494
## y 0.08813303 0.010068663 0.00000000 0 0.000000e+00 0 0.08813303
##              y
## a 6.478782e-05
## c 0.000000e+00
## g 1.306561e-04
## m 0.000000e+00
## r 0.000000e+00
## s 0.000000e+00
## t 0.000000e+00
## y 0.000000e+00
##
## $upperEndpointMatrix
##           a         c          g            m            r            s
## a 0.3444525 0.1778856 0.28097032 0.0001296179 0.0004252881 0.0000000000
## c 0.3881387 0.2547762 0.05328756 0.0000000000 0.0006470811 0.0000000000
## g 0.4018374 0.2089495 0.19912541 0.0001753300 0.0005752738 0.0000000000
## m 0.0000000 0.0000000 1.00000000 0.0000000000 0.0000000000 0.0000000000
## r 0.6677573 0.3440571 0.25448084 0.0000000000 0.0000000000 0.0000000000
## s 0.0000000 1.0000000 0.00000000 0.0000000000 0.0000000000 0.0000000000
## t 0.1557827 0.2174354 0.36484300 0.0001606349 0.0002627497 0.0001606349
## y 0.5785336 0.3899313 0.28841162 0.0000000000 0.0000000000 0.0000000000
##           t            y
## a 0.2189958 0.0004252881
## c 0.3303360 0.0003225840
## g 0.2157436 0.0006648361
## m 0.8816179 0.0000000000
## r 0.5104709 0.0000000000
## s 0.0000000 0.0000000000
## t 0.2868937 0.0002627497
## y 0.5785336 0.0000000000
```

```r
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
```

```r
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. Maybe because we only have 4 nucleotide.

```
#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 y
## a 0.2975779 0.36138614 0.395061728 0.1297710 0
## c 0.1660900 0.24752475 0.205761317 0.2022901 1
## g 0.2698962 0.06435644 0.193415638 0.4007634 0
## t 0.2664360 0.32673267 0.201646091 0.2671756 0
## y 0.0000000 0.00000000 0.004115226 0.0000000 0
##
## $Q[[2]]
##           a         c           g         t y
## a 0.3425606 0.2089552 0.292181070 0.2938931 0
## c 0.1591696 0.1940299 0.222222222 0.2328244 1
## g 0.2733564 0.2587065 0.222222222 0.2213740 0
## t 0.2249135 0.3383085 0.259259259 0.2519084 0
## y 0.0000000 0.0000000 0.004115226 0.0000000 0
##
```

```
##
## $X
##           a           c           g           t           y
## 0.289579158 0.202404810 0.243486974 0.263527054 0.001002004
```

```
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
```

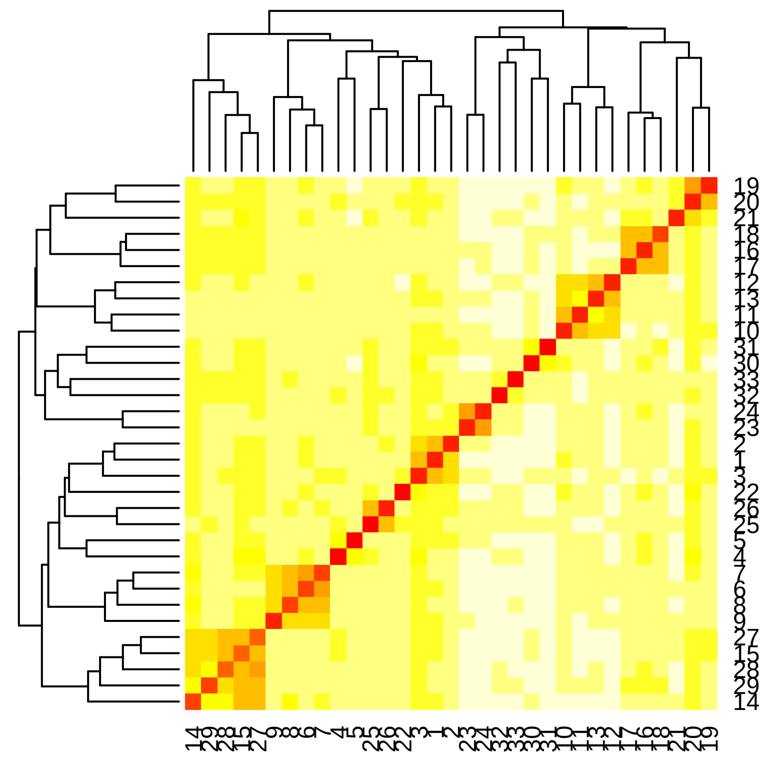
```
real_alignseq<- msaConvert(real_align, type="seqinr::alignment")
dist_real <- as.matrix(dist.alignment(real_alignseq, "identity"))

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

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

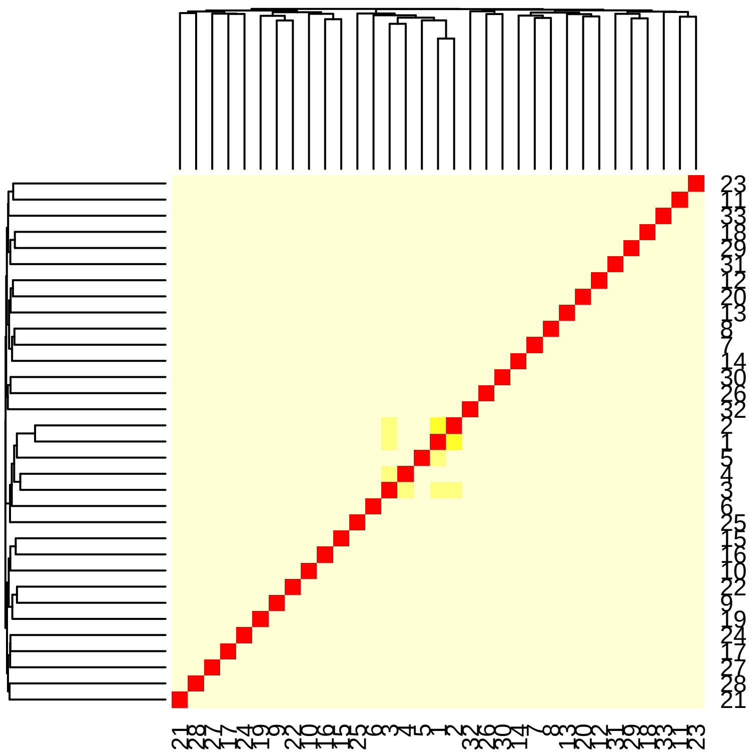
```
ai1.1_alignseq <- msaConvert(ai1.1_align, type="seqinr::alignment")
dist_a1.1 <- as.matrix(dist.alignment(ai1.1_alignseq, "identity"))

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

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

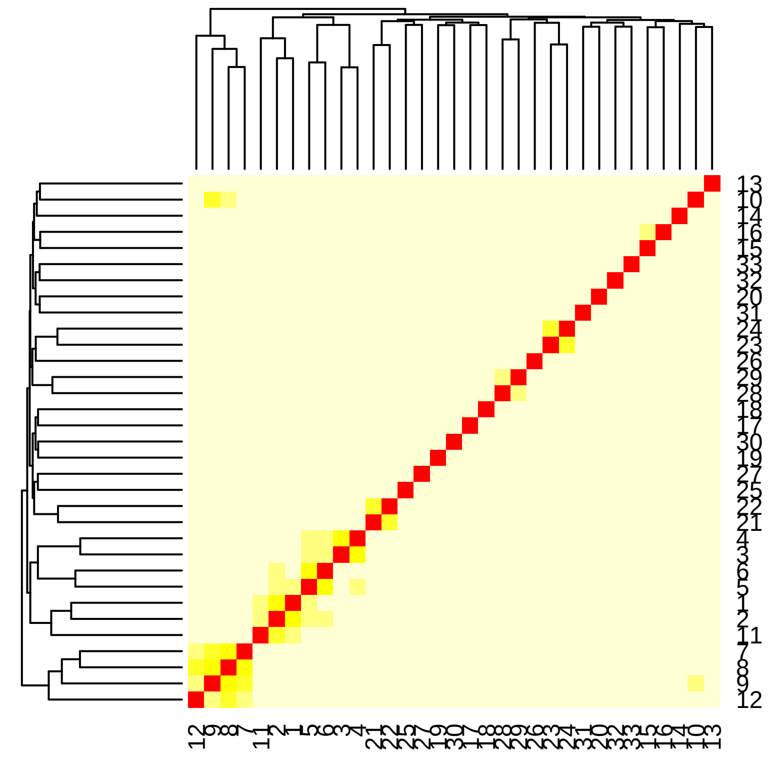
```
ai1.2_alignseq<- msaConvert(ai1.2_align, type="seqinr::alignment")
dist_a1.2 <- as.matrix(dist.alignment(ai1.2_alignseq, "identity"))

heatmap(dist_real)
```



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



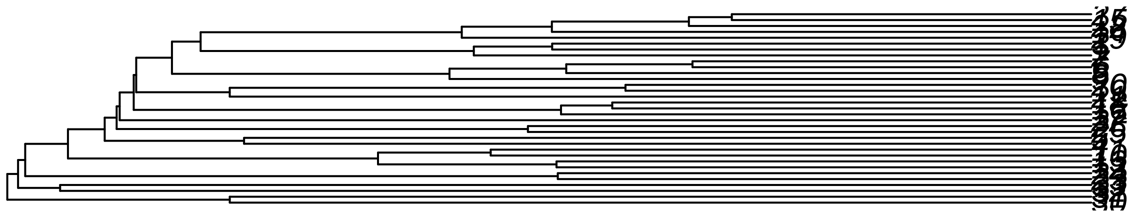
```
heatmap(dist_a1.2)
```



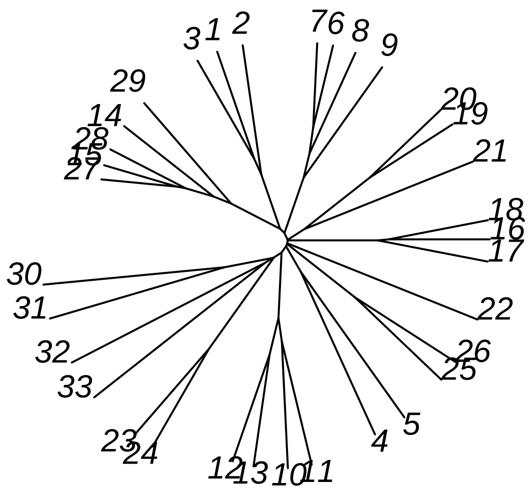
As we can see that the values in heatmap of AI simulated sequences are quite low. It means that the AI simulated gene has low (or no) connections with each other (because it randomly created) while the original gene is highly connected.

Question 3: Phylogeny reconstruction

UPGMA



NJ


3
1
2
7
6
8
9
29
20
19
14
28
15
27
21
18
16
17
30
22
31
32
26
25
33
5
23
24
4
12
13
10
11