Methods and Models in Genomics Assignment

Aleksandr Sahakyan - as952, June 2009

Implementation of the Markov Cluster Algorithm in R

Background

Markov Clustering (MCL) is a powerful algorithm that allows to effectively cluster data [1, 2, 3, 4]. The simple mathematical operations that stay behind the workflow of the program allow an efficient scaling of the memory and CPU usage depending on the problem size, which lets the method to be used in the most demanding applications [1, 4]. To this end, computational biology with its frequently huge data sets becomes one of the major beneficiaries of the MCL algorithm [5].

The whole mathematics behind the technique is just a normal matrix multiplication and taking a Hadamand power (a power of each of the matrix elements, $(m_{ij})^g$) with a certain factor g called granularity. Two matrix multiplication, usually implemented in the algorithm, can be considered as the special case of taking the power of the whole matrix $M = M^k$, where k is frequently taken as 2 [1]. Those two major steps comprise the so called alteration and inflation steps of the MCL workflow, owing to which the gap between the weak and strong connectivities increases.

The method is valid for only stochastic Markov matrices, defined as the matrices for which the sum of each of the columns equals to 1. Thus, the values in the normalized adjacency matrix represent the probabilities of the stochastic flow (connectivity) between the elements i and j [1, 2]. In contrast to the expansion step, the inflation alters the normalized state of the matrix, thus, an additional step of scaling is required at each iteration. Another restraint imposed on the input matrix, is that it should not contain any negative values. The process continues until a convergence reached, which occurs quadratically [1, 2, 3].

Implementation

A program is written under the ***R*** programming and graphics environment [6], to represent the applicability of the MCL algorithm in bioinformatics. The program code is included in the Appendix, and, is additionally deposited in the CamTools site as *as952_MCL.R*. The whole code consists of two main functions - *adjMX.creator* and *MyMCL*.

The *adjMX.creator* function takes as arguments the name of the file with a sequence set in a FASTA format, the location of the *"bin"* folder of local BLAST installation [7] and the type of the search (*blastp* - for proteins, *blastn* - for nucleotide sequences etc.). An example of launching the code in ***R*** is given below:

> Adj.mx.init = adjMX.creator("TEST.fasta","/home/alex/MMG/Program/blast/blast-2.2.20/bin", "blastp")

Obviously, for running the code, the BLAST software should be installed on the system [7]. As a result, the function does all-to-all pairwise sequence homology calculation, being linked to the *bl2seq* subprogram of the local BLAST installation. Then, it reads the output expectation E-values from each of the comparison iterations and stores to a raw matrix, initially converting the read values into $-log_{10}(Evalue)$. The diagonal elements of the matrix correspond to the self alignment, and, of course the E-values are extremely small (therefore $-log_{10}(Evalue)$s are big) because of the 100% sequence identity. Then, the program does several diagnostic and preprocessing steps, to correct possible errors and prepare the matrix for the MCL. In particular, there were several rare cases, when the E-values of the self-comparison was returned as 0, regardless of the 100% sequence identity. Those cases are effectively identified and the numbers are replaced to 1×10^{-200} [5], which correspond to 200 in the minus-logarithmic scale. A single case was noted, when the *bl2seq* calculation output resulted in an output with no E-values reported (although the bit scores and the rest of results were printed). In

that case, the code identifies and assignes pseudo-value of 1, which in $-log$ scale will corrspond to 0 connectivity.

Furthermore, as advised in the reference [5], the maximum cutoff of $-log_{10}(Evalue) = 200$ is implemented, for all the scores which are greater. After these processing actions, the code also symmetrizes the matrix by adding to its transpose and dividing by 2, in order to get rid of possible small differences between the two off-diagonal elements. The mentioned difference was observed while using extremely large data set of all the proteins from the K12 strain of E.coli (used in Genome Informatics module) and was attributed to the cases, when comparisons were done between unrealistically short (7-12 residue) protein segment and a longer one. Finally, the *adjMX.creator* function normalizes column by column the adjacency matrix and returns that, additionally printing some useful information. For further details, please see the thoroughly commented code.

The *MyMCL* function takes as an argument the normalized adjacency matrix and the value of granularity, which should be greater than 1, as the MCL requires [1]. Then, it iterates the algorithm as described in the Background section until the convergence. The convergence is detected as soon as the matrix does not change after an iteration. Finally, the program returns the resulting clustering matrix of 1s and 0s, and creates a graph, representing the clusters. The latter visualization is achieved via the ***R*** function *gplot*, which can be accessible after installation and calling of the *sna* (Social Network Analyser) ***R*** package (see the code).

At the end of the provided code, the written functions are implemented for validation and testing of the results. For farther details, please, refer to the attached source code.

Testing

The testing of the own implementation of the MCL algorithm is done both to validate the biological relevance of the results and to check its full correspondence with the author's original implementation [3]. For this purpose, the following steps are performed.

1. Three proteins are browsed in the Protein Data Bank (PDB) [8] and the corresponding sequences were downloaded in a FASTA format. Those proteins are L-lactate dehydrogenase [9] (PDB accession code is 1LLC), γ-2 aldehyde dehydrogenase [10] (PDB accession code - 1HT0) and ubiquitin [11] (PDB accession code for its NMR structure is 1D3Z).

2. On-line BLAST database search is done via the NCBI server [12] and for each of the above mentioned 3 proteins, 50 closely related homologeous sequences are downloaded in a FASTA format.

3. All those sequences are joined together, to get a single FASTA file, containing the test set for analysis.

Thus, at the end, a set of 150 sequences is included in the *TEST.fasta* file (attached in CamTools), of which 50 are closely related to L-lactate dehydrogenase, other 50 sequences are related to γ-2 aldehyde dehydrogenase and the final 50 to human ubiquitin. Therefore, already knowing the biological and sequence similarities from an on-line BLAST search, we would expect to have 3 clusters of 50 proteins after the application of MCL algorithm.

The results of my MCL application is displayed in Figure 1 as plotted (a) - without the application of the MCL (from the initial normalized adjacency matrix), and (b,c,d) - via the application of my MCL implementation with different granularity values. It is noteworthy to mention, that as we increase the granularity, less iterations are needed for convergence. In case of 2 granularity, the number of required iterations was 21.

As can be inferred from the figure, when an extremaly small 1.1 value of granularity is used, the program just detects one cluster (Figure 1b), and is insensitive towards the differences between the above mentioned three sets of proteins. The granularity value of 2, which is the default one in the original MCL implementation [1], enables to correctly detect the designed subsets of proteins, clearly showing three clusters with the correct (from 1 to 50, from 51 to 100 and from 101 to 150) indices clustered together (Figure 1c). Further increse in the granularity values (an example is at Figure 1d) separates other subsets of small clusters as well.

I also selected two biologically related enzymes (L-lactate dehydrogenase and γ-2 aldehyde dehydrogenase), to see whether their interconnection can be detected by varying the strength of clustering

via the granularity factor in my MCL implementation, but, could not manage to find any interconnection. Further testing on the original MCL program and the examination of the BLAST search showed, that those groups are really unrelated proteins.

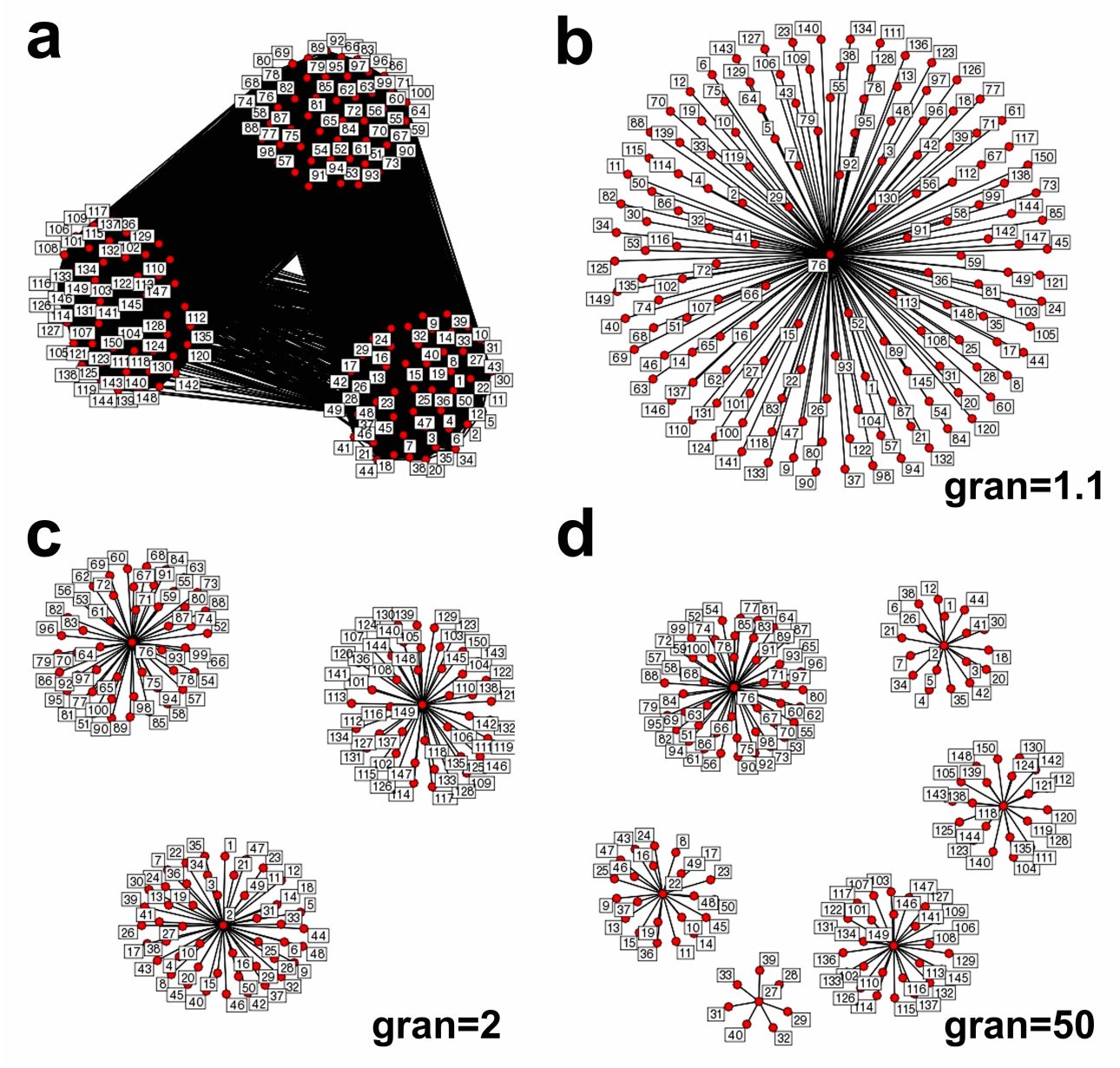

Figure 1: *Interconnection between the 150 protein sequences from the compiled test-set (a) before the application of the MCL algorithm, (b), (c) and (d) after the application of MCL with different granularity values.*

Finally, the comparison of my results with the author's ***C*** implementation of MCL is done (see the final section of the code), installing the original MCL program and its ***R*** interface (*clusterMCL* package), as deposited in the home page of MCL [3]. The resulting matrix clearly demonstrated the presence of 3 clusters, further validating the correctness of own ***R*** code.

P.S. The source code and the files necessary to test the code is deposited in CamTools site.

References

[1] Stijn van Dongen, *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, *http://www.micans.org/mcl/lit/index.html*, 2000.

[2] Stijn van Dongen, *A cluster algorithm for graphs*. Technical Report INS-R0010, National Research Institute for Mathematics and Computer Science in the Netherlands, Amsterdam, May 2000.

[3] Stijn van Dongen, *MCL Program Home Page. http://www.micans.org/mcl/*, accessed in June, 2009.

[4] Lio P., *Methods and Models in Genomics: Lecture Series for Computational Biology MPhil Course*. University of Cambridge, Cambridge, UK, 2009.

[5] Enright A.J., Dongen S.V. and Ouzounis C.A., *An Efficient Algorithm for Large-Scale Detection of Protein Families*. Nucleic Acids Research, vol. 30, pp. 1575-1584, 2002.

[6] R Development Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[7] Altschul S.F., Gish W., Miller W., Myers E.W. and Lipman D.J., *Basic Local Alignment Search Tool*. J. Mol. Biol., vol. 215, pp. 403-410, 1990.

[8] Berman H.M., Westbrook J., Feng Z., Gilliland G., Bhat T.N., Weissig H., Shindyalov I.N. and Bourne P.E., *The Protein Data Bank*. Nucleic Acids Research, vol. 28, pp. 235-242, 2000.

[9] Buehner M. and Hecht H.J., *Structure Determination of the Allosteric L-Lactate Dehydrogenase from Lactobacillus Casei at 3.0 Angstrom Resolution*. Acta Crystallogr. Sect. A, vol. 40, pp. 32-32, 1984.

[10] Niederhut M.S., Gibbons B.J., Perez-Miller S. and Hurley T.D., *Three-Dimensional Structures of the Three Human Class I Alcohol Dehydrogenases*. Protein Science, vol. 10, pp. 697-706, 2001.

[11] Cornilescu G., Marquardt J.L., Ottiger M. and Bax A., *Validation of Protein Structure from Anisotropic Carbonyl Chemical Shifts in a Dilute Liquid Crystalline Phase*. J. Am. Chem. Soc., vol. 120, pp. 6836-6837, 1998.

[12] *NCBI BLAST Server Home Page. http://blast.ncbi.nlm.nih.gov/Blast.cgi*, accessed in June, 2009.

Appendix

The implemented code and all the related files are additionaly deposited in CamTools.

```
##############################################################################
###                  Aleksandr Sahakyan  --  as952                        ###
###                 Methods and Models in Genomics                        ###
###          Code for Clustering With the Markov Algorithm                ###
##############################################################################

####  Input argument examples working on my computer.
#FASTA.name <- "Ecoli.fasta"                  # The name of the fasta file
# The path to the "bin" directory of the local Blast installation.
#blast.bin  <- "/home/alex/MMG/Program/blast/blast-2.2.20/bin"
#search.type <- "blastp"  # The BLAST comparison type to be used.

#gran <- 2.0                           # Granularity for the MCL algorithm.
######################

adjMX.creator <- function(FASTA.name, blast.bin, search.type)  {
```

```
raw.fasta <- readLines(FASTA.name) # reads the fasta file
seq.lines <- grep(">",raw.fasta) # determines the beginings of the seq.
Length <- length(seq.lines)

print(paste("The read FASTA file contains ",Length," sequences.",sep=""))

E.mx <- matrix(NA, nrow=Length, ncol=Length)
Score.mx <- matrix(NA, nrow=Length, ncol=Length)

for(i in 1:Length) {

  print(paste("Analysisng the sequence ",i," ...",sep=""))
  if (i == Length) {   # for the last sequence in a list
    begin.pos <- seq.lines[i]
    end.pos <- length(raw.fasta)
  } else {             # for the rest of situations
    begin.pos <- seq.lines[i]
    end.pos <- seq.lines[i+1] - 1
  }

  single.sequence <- raw.fasta[begin.pos:end.pos]
  # Writing the 1st fasta file with a single sequence
  write(single.sequence, file="1.fasta")

  for(j in 1:Length) {

    if (j == Length) {   # for the last sequence in a list
      begin.pos <- seq.lines[j]
      end.pos <- length(raw.fasta)
    } else {             # for the rest of situations
      begin.pos <- seq.lines[j]
      end.pos <- seq.lines[j+1] - 1
    }

    single.sequence <- raw.fasta[begin.pos:end.pos]
    # Writing the 2nd fasta file with a single sequence
    write(single.sequence, file="2.fasta")

    ####### Doing alignment and extracting the E-values and Score
    # -i  First sequence [File In]
    # -j  Second sequence [File In]
    # -p  Program name: blastp, blastn, blastx
    # -g  Gapped [T/F]
    # -o  alignment output file [File Out]
    # -M  Matrix [String] default = BLOSUM62
    align.command <- paste("'",blast.bin,"/","bl2seq","'"," -p ",
     search.type," -F F -g T  -i 1.fasta -j 2.fasta -o output.out",sep="")
    system(align.command)
    raw.result <- readLines("output.out")
    data.line <- grep("Expect", raw.result)[1]
    if(is.na(data.line)==TRUE)  { # means our data.line is NA
                                  # thus the alignmet did not produce scores
                                  # This is very rare given a normal dataset.
      score <- "NG"  # not generated
      e.value <- 1
    }else{
      data <- raw.result[data.line]
      data <- unlist(strsplit(data," "))
      data <- data[which(data!="")]
      score <- as.numeric(data[3])    #bits
      e.value <- data[8]
```

```
        e.value <- unlist(strsplit(e.value,""))
        if(e.value[1]=="e"){
          e.value[1] <- "1e"
        }
        #Storing num. with correct format
        e.value <- as.numeric(paste(e.value,collapse=""))
        # limiting the lowest possible value to 1e-200
        if(e.value < 1e-200) { e.value <- 1e-200 }
      }
      lnEvalue <- -log10(e.value) ; if(lnEvalue < 0){lnEvalue <- 0}
      E.mx[i,j] <- lnEvalue
      Score.mx[i,j] <- score
    }

  }

  E.mx.raw <- E.mx # Backup-ing the resulting -log matrix

  system("rm 1.fasta 2.fasta output.out") # cleaning the work-space

  # Setting -ln(E) as high, in case of diagonal elements returned as 0
  for(i in 1:Length) { if(E.mx[i,i]==0){E.mx[i,i] <- -log10(1e-200)}}

  # Symmetrizing the matrix
  E.mx <- (E.mx+t(E.mx))/2

  # Converting to a stochastic Markov matrix, so that the sum of each of
  # the columns is 1.
  for(col in 1:Length) { E.mx[,col] <- E.mx[,col]/sum(E.mx[,col]) }

  colnames(E.mx) <- c(1:length(E.mx[1,]))  # Adding colnames (indices)
  rownames(E.mx) <- c(1:length(E.mx[1,]))  # Adding rownames (indices)

  return(E.mx)
}
##############      END OF THE FUNCTION adjMX.creator      ##############


#######################        MCL algorithm         ####################
MyMCL <- function(adjMX, gran) {

Length <- length(adjMX[,1])
adjMX.init <- adjMX  # Backup-ing for gaphing the initial state

for(i in 1:1000) {      # 1000 - number of maximum iteration (usually the
                                  # job converges within 20-25 iterations)
  adjMX1 <- adjMX%*%adjMX # Expansion
  adjMX1 <- adjMX1^gran  # Inflation (1st step)
  for(col in 1:Length){ adjMX1[,col]<-adjMX1[,col]/sum(adjMX1[,col]) } #scaling
  # checking whether the resulting matrix is similar or not
  simil <- sum(as.numeric(as.vector(adjMX==adjMX1))) #calculating similarity btw.
                                                     #the current and previous scores
  if( simil==(Length*Length) ) {
    print("CONVERGED") ; break                # stops the cycle
  } else {
    print("MCL iteration") ; adjMX <- adjMX1
  }
}

print("MCL COMPLETED!!!")
# PLOTTING
```

```
library("sna")  # Loading the necessary library for graphing. In case it
          # does not exist, the following line should be used to install.
          # install.packages("sna")

print("PLEASE WAIT, UNTIL THE PROGRAM FINISHES CREATING THE PICTURES...")

#jpeg(quality=100, height=600, width=600, filename="Initial_graph.jpg")
#  gplot(adjMX.init, gmode = "graph", displaylabels=TRUE)
#dev.off()

jpeg(quality=100, height=600, width=600, filename="Clustered_graph.jpg")
  gplot(adjMX, gmode = "graph", displaylabels=TRUE)
dev.off()

print("FIGURE IS CREATED!!!")

return(adjMX)
}
#####################     END OF FUNCTION MyMCL    ######################


Adj.mx.init <- adjMX.creator("TEST.fasta",
                "/home/alex/MMG/Program/blast/blast-2.2.20/bin", "blastp")
Clust.mx <- MyMCL(adjMX = Adj.mx.init, gran = 2.0)

# Comparison with the original MCL implementation. This bit of the code
# uses the R-interface and the MCL original package installation, as pro-
# vided in the home page of the author.

original <- clusterMCL(Adj.mx.init, granularity=2)
print(original)

print("As can be seen, the original implementation really found 3 clusters")
##########################       THE END       ##########################
```