Chapter 27

Phylogenetic Analyses of Chemokine Receptors from Sequence Retrieval to Phylogenetic Trees

Juan C. Santos

Abstract

Phylogenetic trees are an essential requisite for comparative biology studies where hypotheses regarding the evolution of genes can be investigated. Trees provide visual and statistical guides to characterize the degree of relatedness among biological entities from genes to species. In a tree, ancestor-descendant relationships are represented by connections, and closely related entities share most of these links. In this chapter, I outlined a method to retrieve and label amino acid and nucleotide sequences of chemokine receptors, align them in sequence matrices, determine their best-model of molecular evolution, and estimate the corresponding phylogenetic trees with distance and maximum likelihood approaches. Most of these analyses are performed within the *R* environment, and all of these methods use open-source software.

Key words Amino acids, Nucleotides, Automatic sequence retrieval, Alignment, Homology, Substitution models, Maximum likelihood, Neighbor joining, Phylogenetic trees

1 Introduction

Most comparisons at the level of nucleotide and amino acid sequences should address the evolutionary relationships among the source populations and species before any inferences can be formulated [1]. By accounting for phylogenetic relationships, the characterization of paralogous or orthologous sequences can reveal interesting patterns that can be associated with biological adaptations and key protein functions [2]. However, direct comparisons among sequences can be biased due to phylogenetic signal, which is defined as the similarities between biological entities as a result of inherited resemblances from their ancestors [3]. To avoid such bias, phylogenetic trees are inferred as evolutionary hypotheses of ancestor-descendant relationships derived from sequence alignments, a model of nucleotide or amino acid substitutions and an optimizing algorithm that tries to find the best possible tree structure (i.e., topology). These phylogenetic trees provide information about the degree of divergence (i.e., absolute or relative time)

Ivana Vancurova and Yan Zhu (eds.), *Immune Mediators in Cancer: Methods and Protocols*, Methods in Molecular Biology, vol. 2108, https://doi.org/10.1007/978-1-0716-0247-8_27,

between biological entities since their last common ancestor. By tracing gene sequences on these trees, researchers can infer ancestral states along the nodes of the phylogeny (i.e., hypothesized sequences in ancestors). Furthermore, comparisons among biological entities (e.g., genes, species) can be done using most statistical analyses by adding a correction for the underlying phylogenetic structure.

When reading a phylogenetic tree, its topology (i.e., the relative arrangement of terminal tips and their internal nodes through connecting branches) allows us to infer groupings or clades in which two or more tips (i.e., taxa) are connected to a common ancestor (i.e., a close shared node). These ancestors are inferred to have ancestral phenotypic or molecular states that, in most cases, are now lost due to extinction. The connecting lines between nodes of the tree are referred as branches, and their length represents an estimate of the genetic distance between connecting taxa, tips or clades [4]. If a tree is given a polarity, such phylogeny is said to be rooted by assigning a branch that connects the closest relative to an outgroup outside the group of interest (i.e., ingroup). A tree without polarity is said to be unrooted. The branch length units of a phylogeny can be relative (e.g., sequence divergence) or absolute (e.g., millions of years). Likewise, if the branch lengths (i.e., distances) of a phylogeny are the same from the root to each of the terminal tips, the phylogeny is said to be ultrametric.

Most phylogenetic analyses strive to have fully resolved (i.e., completely bifurcating) trees and well-supported topologies (e.g., high confidence on the presence of connecting nodes) using bootstraps or other measures of accuracy [1]. Toward these goals, the following steps are usually carried out before any phylogenetic tree reconstruction is done. First, a careful collection of sequence data from samples, whose origins can be traced back to their biological source (*see* **Note 1**), should be performed before any analysis. Second, a correct identification or annotation of the sequences should be confirmed to avoid mixing paralogous, pseudogenes or chimeric sequences, so that only orthologous sequences (*see* **Note 2**) or, if the interest is to analyze gene duplications, paralogous sequences are used in any downstream phylogenetic analyses without ambiguity. Third, most sequences include variations in length as result of nucleotide or amino acid insertions and deletions (indels), so a sequence alignment is a prerequisite before the phylogeny estimation. Sequence alignment is, on its own, a computational rigorous process in which gaps (i.e., spaces) are introduced along the sequences in the dataset to maximize homology [5]. With the aligned sequences, a model of molecular substitution is determined, which is necessary for the calculation of the probabilities of change between nucleotides or amino acids along the branches of the phylogeny. Finally, the evolutionary tree can be estimated from the optimal alignment and substitution models.

In sum, a phylogenetic tree represents a visual and statistical representation of the evolutionary diversification of a section of the Tree of Life. Most researchers will find these phylogenetic trees useful, as these hypotheses promote further comparative analyses among biological entities (e.g., sequences, individuals, genes, species). The objective of this chapter is to present commonly used methods to reconstruct molecular phylogenies using chemokine receptors as an example. However, the processes outlined here can be applied to other genes and phenotypic datasets. Likewise, the bulk of data retrieval and management is done through open-source software. For this reason, I will provide a brief introduction about the installation and management of packages in the *R* environment [6]. Then, the actual workflow starts from getting sequences from the NCBI or other public sequence archives, followed by sequence alignment, model estimation of molecular evolution, and phylogenetic tree reconstruction. I would like to emphasize that the software and procedures indicated here are one of many commonly used methodologies based on freely and commercially available programs. When pertinent, I will provide citations for alternative methods and relevant software.

2 Materials

For this chapter, I used a published list of chemokine receptors published by Bachelerie et al. [7]. This list is by no means exhaustive or complete, but it serves to guide the automatic retrieval of those protein and nucleotide sequences from the NCBI database. It is important to note that only sequences with UniProt [8] and NCBI accession numbers can be retrieved if sequence comparisons are requested. I deliberately included the installation of software as part of the Methods section because this chapter, in essence, aims to provide a hands-on practice for users with little or no familiarity with the *R* environment.

3 Methods

3.1 R *Installation*

R is a computing language and free software environment designed for statistical analyses with a graphics interphase [6]. Its computer requirements tend to be minimal, and *R* is distributed as precompiled binary executables. This software is also available in the three main base platforms: Windows, OS X (Mac), and Linux. Most users get *R* from the Comprehensive R Archive Network (CRAN), which hosts the source code, precompiled versions, installation instructions, and many basic manuals:

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

Once *R* is installed in a computer, the user can run the environment in the *R* console, which provides some basic information about the current version of *R* installed, computer platform, and working directory. If no additional packages have been installed, the base library includes essential commands such as reading/writing files, handling databases, standard statistical analyses, plotting, and other environment control functions. There are alternative ways to install and manage *R*, which might be more accessible to certain users (*see* **Note 3**).

3.2 R-Package Management

The *R* environment allows an expansive modularity with >18,000 free software packages available at public repositories including Bioconductor, GitHub, and the Comprehensive R Archive Network (CRAN). The *R* environment provides an easy installation of packages from the web by typing the following function in the *R* console (notice the punctuation, *see* **Note 4**):

```
install.packages("DESIRED_PACKAGE_NAME")
```

The name of the package can be found in the CRAN and a "mirror repository" should be selected. The location of the "mirror repository" should be the physically closest to the user, but most packages can be installed from any "mirror repository" with enough speed. It is important to note that CRAN regularly updates the repository, and it is up to the package developers to keep theirs current. Some functions in *R* packages might become deprecated in newer versions, and some entire packages are archived (i.e., disappear) from CRAN if not maintained by their developers. In those instances, the user can download the desired package using the *R*-package "devtools" [9]:

```
install.packages("devtools")
library(devtools)
devtools::install_version("ggplot2",
 version = "0.9.1",
 repos = "http://cran.us.r-project.org")
```

With "devtools" is also possible to install packages hosted in GitHub that are not currently available in CRAN. GitHub is a web-based hosting site where many software developers deposit working and recent updates of their computer code. Packages can be installed as follows:

```
library(devtools)
install_github("DEVELOPER_GITHUB_NAME/R_PACKAGE_NAME")
```

Some *R* packages used for phylogenomics, sequence retrieval, gene annotation, and gene ontology enrichment can be accessed and downloaded through Bioconductor [10, 11]. To install

packages from this repository, the user can follow these instructions:

```
install.packages("BiocManager")
library(BiocManager)
BiocManager::install("DESIRED_PACKAGE_NAME",
 version = "VERSION_NUMBER")
```

3.3 Installation of R Packages Used for Phylogenetics and Sequence Manipulation

For the methods described here, the user needs to install the following *R*-packages: "annotate" [12], "ape" [13, 14], "phangorn" [15], and "rentrez" [16]:

```
BiocManager::install("annotate") # Using R environments for
annotation
install.packages("ape") # Analyses of Phylogenetics
and Evolution
install.packages("phangorn") # Phylogenetic
Reconstruction and Analysis Package
install.packages("rentrez") # Using R to access the
NCBI eUtils API
```

As noticed above, I added comments (i.e., text not read or run by *R*) preceded from a "#" sign.

The citation of most *R*-packages can be obtained within *R* with few commands (*see* **Note 5**).

3.4 Retrieval of Protein Sequences from Chemokine Receptors Using an Accession Number

For this example, I will work on CXCL chemokine involved in neutrophil trafficking.

1. Retrieve the human CXCL1 protein that has an accession number: P09341 and assign it to an object called "Human_CXCL1" using the *R*-packages "annotate" and "rentrez":

```
library(annotate)
Human_CXCL1 <- annotate::getSEQ("P09341")
[1]
"MARAALSAAPSNPRLLRVALLLLLLVAAGRRAAGASVATELRCQCLQTLQGIHPKNIQSV
NVKSPGPHCAQTEVIATLKNGRKACLNPASPIVKKIIEKMLNSDKSN"

library(rentrez)
Human_CXCL1 <- annotate::entrez_fetch(db = "protein",
 id = "P09341",
 rettype = "fasta")

Human_CXCL1 # to check what is in this object
[1] ">sp|P09341.1|GROA_HUMAN RecName: Full=Growth-regulated
alpha protein; AltName: Full=C-X-C motif chemokine 1;
AltName: Full=GRO-alpha(1-73); AltName: Full=Melanoma
growth stimulatory activity; Short=MGSA; AltName:
Full=Neutrophil-activating protein 3; Short=NAP-3;
```

```
Contains: RecName: Full=GRO-alpha(4-73); Contains: RecName:
Full=GRO-alpha(5-73); Contains: RecName: Full=GRO-alpha
(6-73); Flags:
Precursor\nMARAALSAAPSNPRLLRVALLLLLLVAAGRRAAGASVATELRCQCLQTLQ
GIHPKNIQSVNVKSPGPHCA\nQTEVIATLKNGRKACLNPASPIVKKIIEKMLNSDKSN\n
\n"
```

2. Carry out an automatic BLAST (Basic Local Alignment Search Tool) query [17] using the standard protein search setting (i.e., protein sequence query against the NCBI protein database) using the "BLASTP" option for our given protein:

```
CXCL_df <- annotate::blastSequences
("MARAALSAAPSNPRLLRVALLLLLLVAAGRRAAGASVATELRCQCLQTLQGIHPKNIQS
VNVKSPGPHCAQTEVIATLKNGRKACLNPASPIVKKIIEKMLNSDKSN",
 hitListSize = "500", program =
"blastp",
 timeout=100, as= "data.frame")
```

With these commands, the "blastSequences" function of the *R*-package "annotate" will look for 500 entries (hitListSize = "500"), using "blastp," with a time out of 100 s and return as a data frame object. These results are assigned to an object called "`CXCL_df`" and different columns in this object can be retrieved by their name. To see the names of the columns in this object use the following command:

```
names(CXCL_df)
```

We are interested in the columns with sequences (i.e., `Hsp_qseq`), so these can be obtained by typing `CXCL1_df $Hsp_qseq`. Similarly, we are interested in these sequence annotations, which can be seen by typing `CXCL_df$Hit_def` and `CXCL_df$Hit_id`:

```
CXCL1_df$Hsp_qseq
CXCL_df$Hit_def
CXCL_df$Hit_id
```

3. Construct and write a *.txt file with an annotated data frame that would include species names, accession numbers, and sequences with the retrieved sequences. These files serve as references for later analyses or any other searches that the user might need to perform with those retrieved sequences:

```
CXCL_for_use_df <- subset(CXCL_df, select = c(Hit_def,
Hit_id, Hsp_qseq))
write.table(CXCL_for_use_df, file =
"CXCL_results_500_aminoacids.txt", sep = "\t")
```

4. Read this saved file as follows (the user should make sure that the working directory is the same as the one that contains the *.txt file, *see* **Note 6**):

```
CXCL_for_use_df <-
read.table("CXCL_results_500_aminoacids.txt", header = TRUE,
 sep = "\t", stringsAsFactors = FALSE)
```

5. Construct a file in fasta format that can be used for sequence alignments:

```
CXCL_definitions <- CXCL_for_use_df$Hit_def
extract_names_1_function <- function(x){
sub(".*\\[(.*)\\].*", "\\1", x, perl=TRUE)}
taxa_names_seq <- extract_names_1_function(CXCL_definitions)
taxa_names_seq <- gsub(" ", "_", taxa_names_seq)

CXCL_accession_numbers <- CXCL_for_use_df$Hit_id
extract_names_2_function <-
function(x){sub(".*\\|(.*)\\|.*$", "\\1",
CXCL_accession_numbers, perl=TRUE)}
taxa_accession <-
extract_names_2_function(CXCL_accession_numbers)

taxa_fasta_name <- paste0(">",taxa_names_seq, "_",
taxa_accession)

# construct fasta entries

taxa_fasta_seq_vector <- paste0(taxa_fasta_name, "\n",
CXCL_for_use_df$Hsp_qseq)
```

6. Remove repeated sequences that have the same accession number:

```
taxa_fasta_seq_vector_2 <-
taxa_fasta_seq_vector[!duplicated(taxa_accession)]
taxa_fasta_seq_vector_2 <- taxa_fasta_seq_vector_2
[!is.na(taxa_fasta_seq_vector_2)]
```

7. Remove entries with extraneous annotations (e.g., "synthetic_construct", "Chain_A,"):

```
taxa_fasta_seq_vector_2 <- taxa_fasta_seq_vector_2
[!grepl("synthetic_construct|Chain_A",
taxa_fasta_seq_vector_2)]
```

8. Write a file in fasta format (*see* **Note 7**) with sequences that include the annotation for the species of origin and the corresponding accession numbers:

```
writeLines(taxa_fasta_seq_vector_2, "CXCL_aminoacid.fasta")
```

3.5 Retrieval of Protein Sequences Using a List of Accession Number for Chemokine Receptors

The accession numbers for the chemokine proteins are listed on Table 1 in Bachelerie et al. [7].

1. Retrieve amino acid sequences as vectors using "annotate," by providing a list of accession numbers for human and mouse into data frames:

```
library(annotate)
receptor_names <- c("CXCL1", "CXCL2", "CXCL3", "CXCL4",
"CXCL4L1", "CXCL5", "CXCL6", "CXCL7", "CXCL8", "CXCL9",
"CXCL10", "CXCL11", "CXCL12", "CXCL13", "CXCL14", "Cxcl15",
"CXCL16", "CXCL17", "CCL1", "CCL2", "CCL3", "CCL3L1",
"CCL3L3", "CCL4", "CCL4L1", "CCL4L2", "CCL5", "Ccl6", "CCL7",
"CCL8", "Cc19", "CCL11", "Cc112", "CCL13", "CCL14", "CCL15",
"CCL16", "CCL17", "CCL18", "CCL19", "CCL20", "CCL21", "CCL22",
"CCL23", "CCL24", "CCL25", "CCL26", "CCL27", "CCL28", "XCL1",
"XCL2", "CX3CL1")

human_access_numbers <- c("P09341", "P19875", "P19876",
"P02776", "P10720", "P42830", "P80162", "P02775", "P10145",
"Q07325", "P02778", "O14625", "P48061", "O43927", "O95715",
"NA", "Q9H2A7", "Q6UXB2", "P22362", "P13500", "P10147",
"P16619", "P16619", "P13236", "Q8NHW4", "Q8NHW4", "P13501",
"NA", "P80098", "P80075", "NA", "P51671", "NA", "Q99616",
"Q16627", "Q16663", "O15467", "Q92583", "P55774", "Q99731",
"P78556", "O00585", "O00626", "P55773", "O00175", "O15444",
"Q9Y258", "Q9Y4X3", "Q9NRJ3", "P47992", "Q9UBD3", "P78423")

mouse_access_numbers <- c("P12850", "P10889", "Q6W5C0",
"Q9Z126", "NA", "P50228", "NA", "Q9EQI5", "NA", "P18340",
"P17515", "Q8R392", "P40224", "O55038", "Q6AXC2", "Q9WVL7",
"Q8BSU2", "Q8R3U6", "P10146", "P10148", "P10855", "P10855",
"NA", "P14097", "NA", "NA", "P30882", "P27784", "Q03366",
"Q9Z121", "P51670", "P48298", "Q62401", "NA", "NA", "NA",
"NA", "Q9WUZ6", "NA", "O70460", "O89093", "P84444", "O88430",
"NA", "Q9JKC0", "O35903", "Q5C9Q0", "Q9Z1X0", "Q9JIL2",
"P47993", "NA", "O35188")

human_genes_df <-
data.frame(name=character(),sequence=character(),stringsAs
Factors=FALSE)
mouse_genes_df <-
data.frame(name=character(),sequence=character(),stringsAs
Factors=FALSE)

for(i in 1:length(human_access_numbers)) {
human_gene_name <- paste0(">human_",receptor_names[i],"_",
human_access_numbers[i])
mouse_gene_name <- paste0(">mouse_",receptor_names[i],"_",
```

```
mouse_access_numbers[i])
if(!human_access_numbers[i] == "NA") {
 Sys.sleep(2)
 human_CXCL_name <- human_gene_name
 human_CXCL <-
try(annotate::getSEQ(human_access_numbers[i]), silent =
TRUE)
 if(!class(human_CXCL) == "try-error"){
 one_human_gene_df <-
data.frame(name=human_CXCL_name,sequence=human_CXCL,strings
AsFactors=FALSE)
 human_genes_df <- rbind(human_genes_df,
one_human_gene_df)
 cat("\n retrieved human gene: ", human_CXCL_name, "\n")
 }
 }
if(!mouse_access_numbers[i] == "NA") {
 Sys.sleep(2)
 mouse_CXCL_name <- mouse_gene_name
 mouse_CXCL <-
try(annotate::getSEQ(mouse_access_numbers[i]), silent =
TRUE)
 if(!class(mouse_CXCL) == "try-error"){
 one_mouse_gene_df <-
data.frame(name=mouse_CXCL_name,sequence=mouse_CXCL,strings
AsFactors=FALSE)
 mouse_genes_df <- rbind(mouse_genes_df,
one_mouse_gene_df)
 cat("\n retrieved mouse gene: ", mouse_CXCL_name, "\n")
 }
 }
 }
```

The user should notice that using this approach requires the computer system to pause for 2 s (i.e., `Sys.sleep(2)`) so the automatic retrieval does not cause NCBI servers to return an error.

2. Create a fasta file ready for alignment with the resulting data frame objects:

```
human_sequence_vector <- paste0(human_genes_df$name, "\n",
human_genes_df$sequence)
mouse_sequence_vector <- paste0(mouse_genes_df$name, "\n",
mouse_genes_df$sequence)
human_mouse_sequence_vector <- c(human_sequence_vector,
mouse_sequence_vector)
```

3. Write this file in fasta format (*see* **Note 7**) with sequences that include the annotation for the organism of origin (i.e., mouse or human) and the accession numbers:

```
writeLines(human_mouse_sequence_vector,
"CXC_human_mouse_aminoacid.fasta")
```

3.6 Retrieval of Nucleotide Sequences from Chemokine Receptors

I will continue with the same CXCL1 chemokine example, but I will do a protein sequence query to obtain its matching nucleotide sequence from the NCBI database.

1. Retrieve a nucleotide sequence that corresponds to the human CXCL1 protein, which has the following accession number: P09341. With the retrieved nucleotide sequences, assign them to an object call "Human_CXCL1" in fasta format using "annotate":

```
library(annotate)
CXCL1_nucleotide_df <- annotate::blastSequences
("MARAALSAAPSNPRLLRVALLLLLLVAAGRRAAGASVATELRCQCLQTLQGIHPKNIQS
VNVKSPGPHCAQTEVIATLKNGRKACLNPASPIVKKIIEKMLNSDKSN",
 hitListSize = "20", program = "tblastn",
 timeout=100, as= "data.frame")
```

In this search, we used the "TBLASTN" option (i.e., protein sequence query against the translated NCBI nucleotide database) to obtain nucleotide sequences.

2. From the resulting sequence hits, we see that the object on the "`CXCL1_nucleotide_df`" includes a list of sequences that can be sorted by their highest bit-score (i.e., `CXCL1_nucleotide_df$Hsp_bit-score`). In this case, the best-score nucleotide sequence corresponds to human CXCL1 mRNA transcript with an accession number: BT006880. This sequence can be retrieved following the same procedure for sequence retrieval as illustrated for proteins (see Subheading 3.4):

```
Human_CXCL1_nucleotide <- annotate::getSEQ("BT006880")
Human_CXCL1_nucleotide
[1]
"ATGGCCCGCGCTGCTCTCTCCGCCGCCCCCAGCAATCCCCGGCTCCTGCGAGTGGCACTG
CTGCTCCTGCTCCTGGTAGCCGCTGGCCGGCGCGCAGCAGGAGCGTCCGTGGCCACTG
AACTGCGCTGCCAGTGCTTGCAGACCCTGCAGGGAATTCACCCCAAGAACATCCAA AGTG
TGAACGTGAAGTCCCCCGGACCCCACTGCGCCCAAACCG AAGTCATAGCCACACTCAAG
AATGGGCGGAAAG CTTGCCTCAATCCTGCATCCCCCATAGTTAAGAAAATCATCGAAAA
GATGCTGAACAGTGACAAATCCAACTAG"
```

3. Carry out automatic BLAST queries using the standard nucleotide search setting (i.e., nucleotide sequence query against the NCBI nucleotide database) using the "BLASTN" option for our given nucleotide:

```
CXCL_df <- annotate::blastSequences
("ATGGCCCGCGCTGCTCTCTCCGCCGCCCCCAGCAATCCCCGGCTCCTGCGAGTGGCA
CTGCTGCTCCTGCTCCTGGTAGCCGCTGGCCGGCGCGCAGCAGGAGCGTCCGTGGCCAC
TGAACTGCGCTGCCAGTGCTTGCAGACCCTGCAGGGAATTCACCCCAAGAACATCCAAA
GTGTGAACGTGAAGTCCCCCGGACCCCACTGCGCCCAAACCGAAGTCATAGCCACACTC
AAGAATGGGCGGAAAGCTTGCCTCAATCCTGCATCCCCCATAGTTAAGAAAATCATCGA
AAAGATGCTGAACAGTGACAAATCCAACTAG",
 hitListSize = "500", program =
"blastn",
 timeout=100, as= "data.frame")
```

With these commands, the "blastSequences" function of "annotate" will look for 500 entries (hitListSize = "500"), using "blastn," with a time out of 100 s and return as a data frame object. The retrieved sequences would be on `CXCL1_df $Hsp_qseq` and their associated annotations on `CXCL_df $Hit_def` and `CXCL_df$Hit_id`.

4. Construct and write a *.txt file with an annotated data frame that would include species, accession numbers, and sequences with the retrieved sequences:

```
CXCL_for_use_df <- subset(CXCL_df, select = c(Hit_def,
Hit_id, Hsp_qseq))
write.table(CXCL_for_use_df, file =
"CXCL_nucleotide_df.txt", sep = "\t")
```

5. Read the information of this file back into *R* for later processing with the following commands:

```
CXCL_for_use_df <- read.table("CXCL_nucleotide_df.txt",
header = TRUE,
 sep = "\t", stringsAsFactors = FALSE)
```

6. Construct a file in fasta format that can be used for alignments:

```
CXCL_definitions <- CXCL_for_use_df$Hit_def
extract_names_1_function <-
function(x){sub(".*\\[(.*)\\].*", "\\1", x, perl=TRUE)}
taxa_names_seq <- extract_names_1_function(CXCL_definitions)
taxa_names_seq <- gsub(" ", "_", taxa_names_seq)

CXCL_accession_numbers <- CXCL_for_use_df$Hit_id
extract_names_2_function <-
function(x){sub(".*\\|(.*)\\|.*$", "\\1",
CXCL_accession_numbers, perl=TRUE)}
taxa_accession <-
extract_names_2_function(CXCL_accession_numbers)

taxa_fasta_name <- paste0(">",taxa_names_seq, "_",
```

```
taxa_accession)

# construct fasta entries

taxa_fasta_seq_vector <- paste0(taxa_fasta_name, "\n",
CXCL_for_use_df$Hsp_qseq)
```

7. Remove repeated sequences that have the same accession number:

```
taxa_fasta_seq_vector_2 <-
taxa_fasta_seq_vector[!duplicated(taxa_accession)]
taxa_fasta_seq_vector_2 <- taxa_fasta_seq_vector_2
[!is.na(taxa_fasta_seq_vector_2)]
```

8. Perform sequence clean-up by simplifying entry names (e.g., removing "PREDICTED:_", ",_mRNA"):

```
taxa_fasta_seq_vector_2 <- gsub("PREDICTED:_", "",
taxa_fasta_seq_vector_2, fixed = TRUE)

taxa_fasta_seq_vector_2 <- gsub("-", "",
taxa_fasta_seq_vector_2, fixed = TRUE)
taxa_fasta_seq_vector_2 <- gsub("(", "",
taxa_fasta_seq_vector_2, fixed = TRUE)
taxa_fasta_seq_vector_2 <- gsub(")", "",
taxa_fasta_seq_vector_2, fixed = TRUE)
taxa_fasta_seq_vector_2 <- gsub(",", "",
taxa_fasta_seq_vector_2, fixed = TRUE)
taxa_fasta_seq_vector_2 <- gsub("mRNA", "",
taxa_fasta_seq_vector_2, fixed = TRUE)
taxa_fasta_seq_vector_2 <- gsub("__", "_",
taxa_fasta_seq_vector_2, fixed = TRUE)
```

9. Remove entries with extraneous annotations (e.g., "Synthetic_construct"):

```
taxa_fasta_seq_vector_3 <-
taxa_fasta_seq_vector_2[!grepl("Synthetic_construct",
taxa_fasta_seq_vector_2)]
```

10. Write the final file in fasta format with species names and accession numbers for sequence alignments:

```
writeLines(taxa_fasta_seq_vector_3,
"CXCL_nucletotide_out.fasta")
```

These files can be modified by the user using a text editor, if further annotations should be incorporated.

3.7 Alignment of Amino Acid or Nucleotide Sequences from Chemokine Receptors

The files that contain the gene sequences in fasta format need to be aligned before the phylogeny can be estimated. Alignment is an iterative process that introduces gaps or spaces so each position along the matrix of sequences represent a homologous character (i.e., same nucleotide or amino acid position). This process can be achieved using sequence aligners that include command, online and GUI-based programs. We will use CLUSTAL OMEGA (online) [18, 19] and SeaView (GUI-based) [20]. For this example, we will use these files: "`CXC_human_mouse_aminoacid.fasta`," "`CXCL_nucletotide_out.fasta`," and "`CXCL_diversity_nucleotide_human.txt`." This latter file was created using the same procedure for nucleotide retrieval (*see* Subheading 3.6).

3.7.1 CLUSTAL OMEGA

To use this web application, the user needs to follow these steps.

1. Using a web browser, access the CLUSTAL OMEGA page: https://www.ebi.ac.uk/Tools/msa/clustalo/ and select the type of sequence format (i.e., PROTEIN, DNA or RNA).
2. The sequences could be pasted or uploaded. In our example, we paste directly the sequences into the web application.
3. Select "Pearson/FASTA" as the output format. The fasta format will preserve the long names associated with our sequences.
4. We have the option to provide an email contact to receive a file with the aligned sequences. In this case, we would click on submit and wait until the sequences are aligned.
5. After the alignment process is finished, check and download the aligned sequences. In this step, a new window will appear and we will copy and paste our alignment in a text document. Save this file with a pertinent name, e.g., `CXC_human_mouse_aminoacid_aligned_CLUSTAL.fasta`.

3.7.2 SeaView

This software is an open-source multiplatform GUI program for molecular phylogeny analyses and alignment [20]. It is provided freely and the user needs to download, install, and run the program with these steps:

1. SeaView is available from: http://doua.prabi.fr/software/seaview. This program can run in many platforms including: OS X (Mac), Windows, and Linux; SeaView source code is also available. The user should choose the platform that corresponds to their computer.
2. Once running, Seaview allows user to "drop" their unaligned sequences (e.g., fasta or txt files) and then the alignment function can be call.
3. Seaview can use different aligners including CLUSTAL OMEGA (default) or MUSCLE [21]. In this exercise, we use

MUSCLE by changing the option from the tab menu: Align > Alignment Options > muscle.

4. Start the alignment algorithm by selecting from the tab menu: Align > Align All.
5. After the alignment is completed, it can be inspected visually and saved as fasta file (i.e., *.fst).

For a detailed use of SeaView, the users are referred to its manual and website.

3.8 Determining the Best Model of Molecular Evolution for Aligned Chemokine Receptors

For this procedure, we will use the files: "`CXC_human_mouse_aminoacid_CLUSTAL_aligned.fasta`" and "`CXCL_diversity_nucleotide_human_MUSCLE.txt`." We will not use the file: "`CXCL_nucletotide_CLUSTAL_aligned.fasta`" because this alignment showed little diversity to do any meaningful phylogenetic tree inference.

3.8.1 Amino Acid Alignments

We would estimate the substitution models for the amino acid alignments while estimating a phylogenetic tree using RAxML 8.2.9 (Randomized Axelerated Maximum Likelihood) [22]. These procedures are provided in Subheading 3.9.1.

3.8.2 Nucleotide Alignments

For these type of data, we will use jModelTest2 [23]. This program provides an extensive statistical selection of models of nucleotide substitution, which are usually required to estimate phylogenetic trees. The software is provided freely and the user needs to download and install the program. For detailed use of this software, the users are referred to the jModelTest2 manual.

1. This program is available at: https://github.com/ddarriba/jmodeltest2. This program is written in Java and it can be used in most platforms that allow to execute a Java Runtime Environment (JRE). As indicated by the authors, jModelTest2 also depends on third-party binaries (e.g., PhyML [24]). In most cases, these software dependencies are installed with jModelTest2.
2. To execute the program, the user should start the java session by clicking on the file "jModelTest.jar." The alignment file can be in fasta format, which is loaded by clicking on the tab menu: File > Load DNA alignment. We can choose the location of our file (e.g., `CXCL_diversity_nucleotide_human_MUSCLE.txt`) and, if successful, the jModelTest2 console will show the number of sequences loaded and the number nucleotide sites.
3. Select the number of models to test and computer processors to be used during model evaluation. The user should click on the tab menu: Analysis > Compute likelihood scores. An interactive menu will provide with options on the number of

processors and likelihood settings. In most cases, the default options are recommended (see the jModelTest2 manual for details). To run model analyses, the user needs to click on "Compute Likelihoods." A new pop-up window will appear indicating the models being computed in real time.

4. After the model evaluations are done, compute the best model by selecting a model comparison method: Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), or performance-based decision theory (DT) calculations. For this exercise, we select AIC by clicking on the tab menu: Analysis > Do AIC calculations and selecting AICc (click on "Use AIC correction") and then on "Do AICc calculations." The results would be presented in the jModelTest2 console and those can be copied to a text file.
5. Explore the output and search for "Model selected." For our alignment file, the model selected was TrN+G (Tamura and Nei [25] with gamma rate variation) with the corresponding partition of "010020." In the output, we can also see some empirical values for the gamma parameter (i.e., G or gamma shape = 2.1300) and base frequencies: freqA = 0.2620, freqC = 0.2842, freqG = 0.2438 and freqT = 0.2099). The user should notice that these base frequencies are not equal (i.e., ~0.25).

3.9 Phylogenetic Inference Using Aligned Sequences in RAxML

The core of most phylogenetic procedures consists in the estimation of the phylogenetic tree and obtaining the support values for the nodes on that tree topology. In this exercise, we will use a maximum likelihood (ML) approach with a commonly used program: RAxML 8.2.9 [22]. Most programs that infer phylogenetic trees require installation. The instructions to do such installations and computer requirements are indicated on the RAxML website: https://github.com/stamatak/standard-RAxML.

3.9.1 Trees Inferred from Amino Acid Alignments

For this analysis, we will follow this procedure:

1. Run RAxML after compiling the software that uses the multicore processor architecture, which is common in most computers (see RAxML manual and website for details). To run the phylogenetic reconstruction, place the file with aligned sequences in the same folder with the compiled RAxML program. Here, I provide an example for this program execution:

```
raxmlHPC-PTHREADS-SSE3 -T 2 -m PROTGAMMAAUTO -N 3 -s
MY_ALIG.fasta -p 12345 -n TEST
```

These commands include the following arguments: -T corresponds to the number of threads that the user want to run and it should be at least 2, but no more than the physical cores of the computer. The -m argument corresponds to the

model of substitution. In this case, we chose "PROTGAMMAAUTO" that indicates that the best-fitting model for amino acid substitution would be selected automatically while the tree estimation is running using a Bayesian Information Criterion (BIC). The -N argument corresponds to the number of alternative runs on distinct starting trees. The -s argument corresponds to the name of the alignment file in fasta format. The -p argument corresponds to any random number seed for the parsimony inferences. The argument -n corresponds to the name of the output file. For OS X (Mac), a "terminal" console is opened (for Windows users, *see* **Note 8**) to run the RAxML analyses with the following commands:

```
./raxmlHPC-PTHREADS-SSE3 -T 10 -m PROTGAMMAAUTO -N 3 -s
CXC_human_mouse_aminoacid_aligned_MUSCLE.fasta -p 12345 -n
CXC_raxml_trees
```

After the analysis is finished, the user can read the information about the model of amino acid substitution selected for our protein alignment, e.g., in this case the model was "JTT" [26]. The program will also output the information about the run, the location of the inferred repetitions, and the best -scoring ML tree. The format of the trees is "newick" which can be visualized later (see Subheading 3.11).

2. Estimate the support of nodes in this ML phylogeny by running 100 bootstraps. To run this analysis, we add the argument -b with a random seed number (e.g., 123476), changing the number of replicates to -N 100, and the name of the output file -n CXC_raxml_boot_trees. We run again these commands in a "terminal" console:

```
./raxmlHPC-PTHREADS-SSE3 -T 10 -b 123476 -m PROTGAMMAAUTO -N
100 -s CXC_human_mouse_aminoacid_aligned_MUSCLE.fasta -p 12345
-n CXC_raxml_boot_trees
```

These boostrap trees lack branch lengths, but the user can change this option (*see* **Note 9**).

3.9.2 Trees Inferred from Nucleotide Alignments

For this analysis, we will follow this procedure.

1. As indicated for amino acid sequences, run RAxML after compiling the software to allow the use of multicore processors. To run the analyses with our aligned nucleotide sequences, we placed the file with sequences in the same folder with compiled RAxML application. An example of the commands is the following:

```
raxmlHPC-PTHREADS-SSE3 -T 2 -m GTRGAMMA -N 3 -s
MY_ALIG.fasta -p 12345 -n TEST
```

These commands include the following arguments: -T corresponds to the number of threads that the user want to run and it should be at least 2, but no more than the physical cores of the computer. The -m argument corresponds to the model of nucleotide substitution. In this case, "GTRGAMMA" indicates one of the many models supported by this software (see RAxML manual for information about models). The -N argument corresponds to the number of alternative runs on distinct starting trees. The -s argument corresponds to the name of the nucleotide alignment file in fasta format. The -p argument corresponds to any random number seed for the parsimony inferences. The argument -n corresponds to the name of the output file. For OS X (Mac), the RAxML analysis is run (for Windows users, *see* **Note 8**) with the following commands:

```
./raxmlHPC-PTHREADS-SSE3 -T 10 -m GTRGAMMA -N 10 -s
CXCL_diversity_nucleotide_human_MUSCLE.fasta -p 12345 -n
CXC_nucleotide_raxml_trees
```

The users should notice that we chose "GTRGAMMA," which corresponds to GTR + Optimization of substitution rates + GAMMA rate heterogeneity. This model is more complex than the TrN+G selected by jModelTest2, but GTR+G approximates well TrN+G. However, the user can override complex models by using arguments such as "--JC69," "--K80," or "--HKY85" (see RAxML manual for information). The program will also output the information about the run, the location of the inferred repetitions, and the best -scoring ML tree. The format of this tree is "newick" which can be visualized later (see Subheading 3.11).

2. Estimate the support of nodes in this ML phylogeny by running 100 bootstraps. To run this analysis, we add the argument -b with a random seed number (e.g., 123476), changing the number of replicates to -N 100, and the name of the output file -n CXC_nucleotide_raxml_boot_trees. We run again these commands in a "terminal" console:

```
./raxmlHPC-PTHREADS-SSE3 -T 10 -b 123476 -m GTRGAMMA -N 100
-s CXCL_diversity_nucleotide_human_MUSCLE.fasta -p 12345 -n
CXC_nucleotide_raxml_boot_trees
```

These boostrap trees lack branch lengths, but the user can change this option (*see* **Note 9**).

3.10 Phylogenetic Inference Using Aligned Sequences in R

We can also estimate phylogenetic trees within the *R* environment. For this purpose, we can use the *R*-packages "ape" [13, 14] and "phangorn" [15]. This last package provides a series of tools to estimate trees from amino acid and nucleotide alignments. All these procedures start by reading the alignment file:

```
library(ape)
library(phangorn)

CXC_aminoacid <- phangorn::read.phyDat(file =
"CXC_human_mouse_aminoacid_aligned_MUSCLE.fasta",
 type ="AA",
 format = "fasta")

CXC_nucleotide <- phangorn::read.phyDat(file =
"CXCL_diversity_nucleotide_human_MUSCLE.fasta",
 type ="DNA",
 format = "fasta")
```

3.10.1 Trees Inferred Using Distance-Based Methods

For this type of trees, we need to infer a matrix of distances and the model that best fits our data. From the previous sections (see Subheading 3.9), we found that the CXC amino acid data were best fit with a JTT model, while the nucleotide data were best fit with a TrN+G model. For distance-based methods, "phangorn" allows many amino acid models including JTT, but only two for nucleotide models: "JC69" for equal base frequencies [27] and "F81" for empirical base frequencies [28]. Based on the output of jModelTest2, we found that the frequency of bases was unequal suggesting that the model "F81" is preferred. Likewise, we also know that the gamma parameter (i.e., the G in TrN+G was 2.1300) and this can be added with the argument "shape = 2.13." To estimate these distances-based trees, we will use the Neighbor Joining [29] method that results in unrooted trees:

```
distance_CXC_aminoacid <- phangorn::dist.ml(CXC_aminoacid,
model = "JTT")
distance_CXC_nucleotide <- phangorn::dist.ml(CXC_nucleotide,
model = "F81", shape = 2.13)

NJ_tree_CXC_aminoacid <-
phangorn::NJ(distance_CXC_aminoacid)
NJ_tree_CXC_nucleotide <-
phangorn::NJ(distance_CXC_nucleotide)

# save these trees to files

ape::write.tree(NJ_tree_CXC_aminoacid, file =
"NJ_tree_CXC_aminoacid.tree")
ape::write.tree(NJ_tree_CXC_nucleotide, file =
"NJ_tree_CXC_nucleotide.tree")
```

3.10.2 Trees Inferred Using Maximum Likelihood

We can also optimize an input tree using a maximum likelihood approach in *R*. For this purpose, we assign an object that will include the alignment, a starting tree, and the parameters for the optimization using ML. For this initial tree, we can use the NJ tree inferred on the previous section with the following commands:

```
##### For the CXC amino acid alignment

fit_CXC_aminoacid <- phangorn::pml(NJ_tree_CXC_aminoacid,
data = CXC_aminoacid)
fitJTT <- update(fit_CXC_aminoacid, model = "JTT", k = 4,
inv = 0.2)
fitJTT_opt <- phangorn::optim.pml(fitJTT ,
 optInv = TRUE,
 optGamma = TRUE,
 rearrangement = "stochastic",
 control = pml.control(trace = 0))

# we can see some of the parameters associated with this
tree

fitJTT_opt

# loglikelihood: -18278.72

# unconstrained loglikelihood: -2786.529
# Proportion of invariant sites: 3.465281e-05
# Discrete gamma model
# Number of rate categories: 4
# Shape parameter: 2.570192
# Rate matrix:

# save this tree to file

ape::write.tree(fitJTT_opt$tree, file =
"JTT_tree_CXC_aminoacid.tree")

# calculate 100 bootstraps and save file

fitJTT_opt_bs <- phangorn::bootstrap.pml(fitJTT_opt,
bs=100, optNni=TRUE, multicore=TRUE)
ape::write.tree(fitJTT_opt_bs, file =
"JTT_tree_CXC_aminoacid_boot.tree")

### For the CXC nucleotide alignment

fit_CXC_nucleotide <- phangorn::pml(NJ_tree_CXC_nucleotide,
data = CXC_nucleotide)
fitTrNG <- update(fit_CXC_nucleotide, k=4)
fitTrNG_opt <- phangorn::optim.pml(fitTrNG,
 model = "TrN",
 optInv = FALSE,
 optGamma = TRUE,
 rearrangement = "stochastic",
```

```
 control = pml.control(trace = 0))

### we can see some of the parameters associated with this tree

fitTrNG_opt

# loglikelihood: -6621.57

# unconstrained loglikelihood: -5050.672
# Discrete gamma model
# Number of rate categories: 4
# Shape parameter: 2.060177

# Rate matrix:
# a c g t
# a 0.000000 1.000000 3.420504 1.000000
# c 1.000000 0.000000 1.000000 2.416139
# g 3.420504 1.000000 0.000000 1.000000
# t 1.000000 2.416139 1.000000 0.000000

# Base frequencies:
# 0.2611841 0.2863102 0.2431908 0.209315

### save this tree to file

ape::write.tree(fitTrNG_opt$tree, file =
"TrNG_tree_CXC_nucleotide.tree")

### calculate 100 bootstraps and save file

fitTrNG_opt_bs  <-  phangorn::bootstrap.pml(fitTrNG_opt,
bs=100, optNni=TRUE, multicore=TRUE)
ape::write.tree(fitTrNG_opt_bs, file =
"TrNG_tree_CXC_nucleotide_boot.tree")
```

For windows users, the "multicore" argument of the function "bootstrap.pml" cannot be used (*see* **Note 10**). Likewise, the values for bootstraps derived from "phangorn::bootstrap.pml" should be taken with caution (*see* **Note 11**).

3.11 Visualizing Phylogenetic Trees

The last part of most phylogenetic analyses is the visualization of the resulting phylogenies and preparing figures as *.jpg or *.pdf. There is many GUI visualization software (*see* **Note 12**), but the *R*-environment provides enormous flexibility in phylogenetic trees display and manipulation. For this reason, I included a simple

approach using the "ape" and "phangorn" *R*-packages to print and save trees.

1. Plot the phylogenies inferred from amino acid alignments with the following code:

```
#### Read RAxML trees in R and assign them to objects:

human_mouse_CXC_tree <-
ape::read.tree("RAxML_bestTree.CXC_raxml_trees")
human_mouse_CXC_boostrap_trees <-
ape::read.tree("RAxML_bootstrap.CXC_raxml_boot_trees")

# best tree with bootstrap values:

phangorn::plotBS(tree = midpoint(human_mouse_CXC_tree),
 BStrees = human_mouse_CXC_boostrap_trees,
 p = 50,
 type = "phylogram",
 bs.col = "blue",
 cex = 0.9)
title("RAxML 100 bootstrap replicates based on aligned CXC
amino acid")
add.scale.bar(cex = 0.7, font = 2, col = "red")

#### Read NJ tree in R and assign them to objects:

human_mouse_CXC_NJ_tree <-
ape::read.tree("NJ_tree_CXC_aminoacid.tree")
plot(human_mouse_CXC_NJ_tree, type = "unrooted", main=" NJ
CXC aminoacid", cex = 0.9)
add.scale.bar(cex = 0.7, font = 2, col = "red")

#### Read ML tree in R and assign them to objects:

human_mouse_CXC_ML_tree <-
ape::read.tree("JTT_tree_CXC_aminoacid.tree")
human_mouse_CXC_ML_boot_tree <-
ape::read.tree("JTT_tree_CXC_aminoacid_boot.tree")

# best tree with bootstrap values:

phangorn::plotBS(tree = midpoint(human_mouse_CXC_ML_tree),
 BStrees = human_mouse_CXC_ML_boot_tree,
 p = 50,
 type = "phylogram",
 bs.col = "blue",
 cex = 0.9)
title("ML 100 bootstrap replicates based on aligned CXC
```

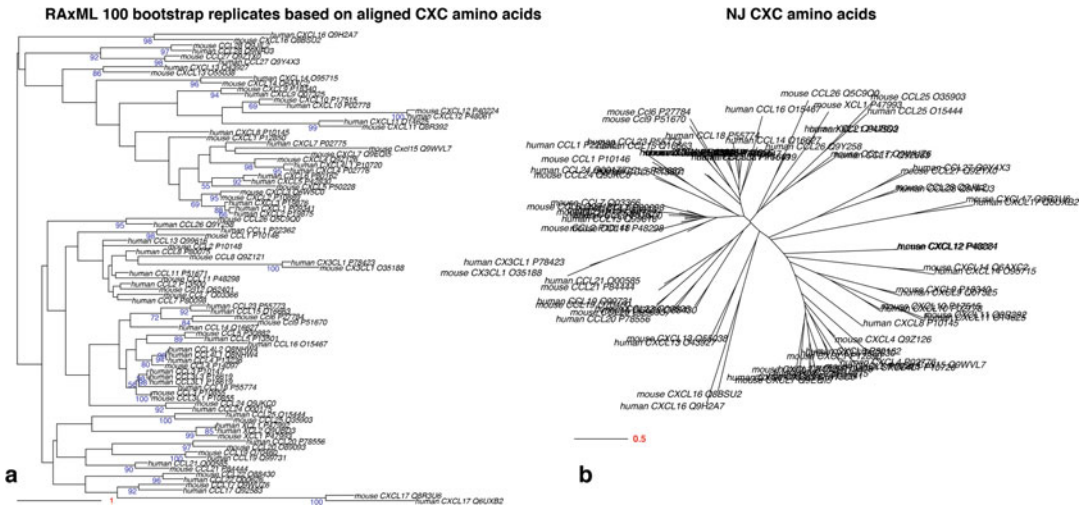

Fig. 1 Phylogeny of chemokine proteins from human and mouse as listed on Table 1 from Bachelerie et al. [7]. (**a**) Maximum Likelihood (ML) phylogeny inferred from the amino acid sequences aligned using MUSCLE with RAxML. The values at the nodes are ML bootstrap percentages $\geq$50%. (**b**) Neighbor Joining (NJ) phylogeny inferred from the amino acid sequences aligned using MUSCLE with the "phangorn" *R*-package

```
amino acid")
add.scale.bar(cex = 0.7, font = 2, col = "red")
```

These commands will allow the user to see the image of the trees (Fig. 1). The following arguments of the function "plotBS" of "phangorn" are indicated: "tree" indicates the phylogenetic tree on which edges the bootstrap values would be plotted (note: this tree is unrooted, so the "midpoint" function was used to perform a midpoint rooting); "BStrees" indicates the object with a list of trees that would be used to estimate the boostrap values; "p" indicates the minimal support percentage that would be plotted on the tree and values > 75 for ML trees are usually considered meaningful; "type" indicates the tree configuration to plot (i.e., "cladogram", "phylogram", or "unrooted"); "bs.col" indicates the color of bootstrap support labels; and "cex" indicates the relative size of the label font. The user can get more information about the function "plotBS" by typing: ?plotBS (*see* **Note 13**). We also added a title to the plot with the function "title" and a scale bar with "add.scale.bar."

We can save the plot image to a *.pdf file with the following commands:

```
# Open a pdf file and save tree plot:

pdf("human_mouse_CXC_boostrap_aminoacid_tree.pdf",
```

```
width=10, height=10)

phangorn::plotBS(midpoint(human_mouse_CXC_tree),
 human_mouse_CXC_boostrap_trees,
 p = 50,
 type = "phylogram",
 bs.col = "blue",
 cex = 0.9)
title("RAxML 100 bootstrap replicates based on aligned CXC
amino acid")
add.scale.bar(cex = 0.7, font = 2, col = "red")

plot(human_mouse_CXC_NJ_tree, type = "unrooted", main=" NJ
CXC aminoacid", cex = 0.9)
add.scale.bar(cex = 0.7, font = 2, col = "red")

phangorn::plotBS(tree = midpoint(human_mouse_CXC_ML_tree),
 BStrees = human_mouse_CXC_ML_boot_tree,
 p = 50,
 type = "phylogram",
 bs.col = "blue",
 cex = 0.9)
title("ML 100 bootstrap replicates based on aligned CXC
amino acid")
add.scale.bar(cex = 0.7, font = 2, col = "red")

dev.off()
```

2. With a similar procedure, plot the trees based on the nucleotide alignments (Fig. 2) with the following code:

```
#### Read RAxML trees in R and assign them to objects:

nucleotide_CXC_tree <-
ape::read.tree("RAxML_bestTree.CXC_nucleotide_raxml_trees")
nucleotide_CXC_boostrap_trees <-
ape::read.tree("RAxML_bootstrap.CXC_nucleotide_raxml_boot_
trees")

# best tree with bootstrap values:

phangorn::plotBS(tree = midpoint(nucleotide_CXC_tree),
 BStrees = nucleotide_CXC_boostrap_trees,
 p = 50,
 type = "phylogram",
 bs.col = "blue",
 cex = 0.9)
title("RAxML 100 bootstrap replicates based on aligned CXCL
nucleotide sequences")
```

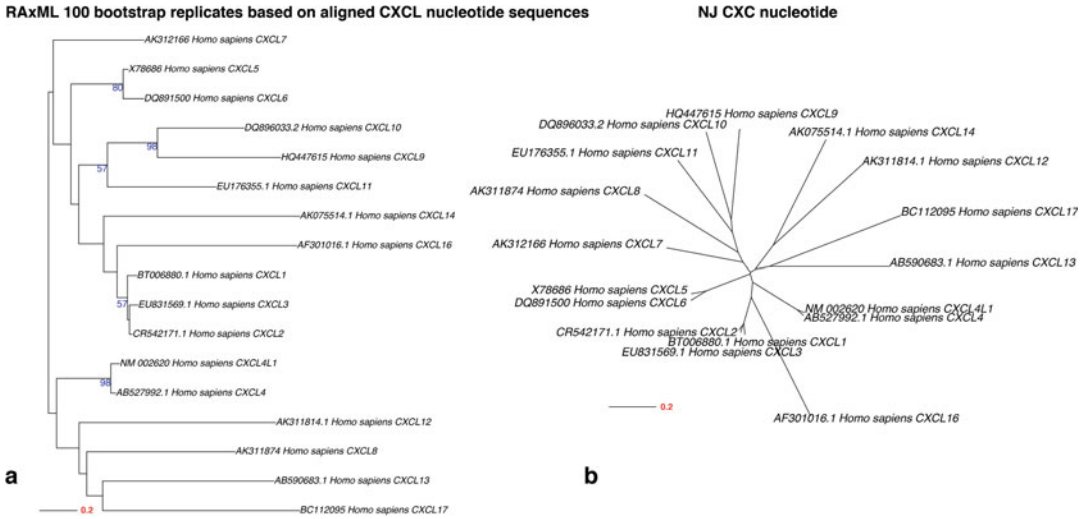

Fig. 2 Phylogeny of CXCL proteins from human specimens. (**a**) Maximum Likelihood (ML) phylogeny inferred from the nucleotide sequences aligned using MUSCLE with RAxML. The values at the nodes are ML bootstrap percentages $\geq$50%. (**b**) Neighbor Joining (NJ) phylogeny inferred from the nucleotide sequences aligned using MUSCLE with the "phangorn" *R*-package

```
add.scale.bar(cex = 0.7, font = 2, col = "red")

#### Read NJ tree in R and assign them to objects:

nucleotide_CXC_NJ_tree <- ape::read.tree("NJ_tree_CXC_nucleo
tide.tree")
plot(nucleotide_CXC_NJ_tree, type = "unrooted", main=" NJ CXC
nucleotide", cex = 0.9)
add.scale.bar(cex = 0.7, font = 2, col = "red")

#### Read ML tree in R and assign them to objects:

nucleotide_CXC_ML_tree <-
ape::read.tree("TrNG_tree_CXC_nucleotide.tree")
nucleotide_CXC_ML_boot_tree <-
ape::read.tree("TrNG_tree_CXC_nucleotide_boot.tree")

# best tree with bootstrap values:

phangorn::plotBS(tree = midpoint(nucleotide_CXC_ML_tree),
 BStrees = nucleotide_CXC_ML_boot_tree,
 p = 50,
 type = "phylogram",
 bs.col = "blue",
 cex = 0.9)
title("ML 100 bootstrap replicates based on aligned CXC
nucleotide")
```

```
add.scale.bar(cex = 0.7, font = 2, col = "red")

### Open a pdf file and save tree plot:

pdf("human_CXCL_nucleotide_boostrap_tree.pdf",  width=10,
height=10)
phangorn::plotBS(tree = midpoint(nucleotide_CXC_tree),
 BStrees = nucleotide_CXC_boostrap_trees,
 p = 50,
 type = "phylogram",
 bs.col = "blue",
 cex = 0.9)
title("RAxML 100 bootstrap replicates based on aligned CXCL
nucleotide sequences")
add.scale.bar(cex = 0.7, font = 2, col = "red")

plot(nucleotide_CXC_NJ_tree, type = "unrooted", main=" NJ
CXC nucleotide", cex = 0.9)
add.scale.bar(cex = 0.7, font = 2, col = "red")

phangorn::plotBS(tree = midpoint(nucleotide_CXC_ML_tree),
 BStrees = nucleotide_CXC_ML_boot_tree,
 p = 50,
 type = "phylogram",
 bs.col = "blue",
 cex = 0.9)
title("ML 100 bootstrap replicates based on aligned CXC
nucleotide")
add.scale.bar(cex = 0.7, font = 2, col = "red")
dev.off()
```

The resulting pdfs with the tree plots can be seen in the working directory (*see* **Note 6**). The user is encouraged to explore more and change parameters for better tree inferences and presentation (*see* **Note 14**).

4 Notes

1. Most errors in phylogenetic estimations are due to mistakes derived from human oversights, which, in turn, derive from mislabeled samples, chimeric sequences, and poor-quality input data. All researchers should spend time to check the quality of their input data (if these sequences derive from PCR products, Next-Gen experiments or other sources). An easy and quick approach is to "BLAST" (https://blast.ncbi.nlm.nih.gov/Blast.cgi) the unknown sequence against the NCBI database. Most times, this sequence will match a sequence already present and the annotation or source can be verified.

2. Orthologous genes are those makers whose similarities (i.e., homology) are the result of speciation so that the history of the gene reflects the history of the species being compared. Most phylogenetic analyses are derived from homologous markers, whose sequences have been aligned to maximize homology at the level of nucleotide or amino acid site positions. If some parts of the alignment are "unalignable" or ambiguous, the user can exclude such sites by deleting or excluding those parts from the final alignment matrix before the tree estimation.
3. Some users find that the interaction with *R* console is cumbersome. In this case, *RStudio* constitutes an alternative software for integrated development and management of the environment for *R*. The desktop version of *RStudio* is free and can be found here:

```
https://www.rstudio.com/products/RStudio/
```

4. The punctuation in coding and writing scripts is very important. The use of quotation marks [""], periods [.], colons [:], and spaces is commonly overseen by first-time users of *R*. One example is the use of curvy quotation marks [“”] instead of straight quotation marks [""]. *R* will only accept straight quotation marks, while it will throw errors if curvy quotation marks [“”] are used instead.
5. To cite *R*-packages, libraries need to be installed and then loaded in the *R* environment as follows:

```
install.packages("devtools")
library(devtools)
citation("devtools") # a function for getting the
information about citation

To cite package ‘devtools’ in publications use:

 Hadley Wickham, Jim Hester and Winston Chang (2019). devtools:
Tools to Make Developing R Packages Easier. R
package
 version 2.0.2. https://CRAN.R-project.org/package=devtools

A BibTeX entry for LaTeX users is

 @Manual{,
  title = {devtools: Tools to Make Developing R Packages
Easier},
 author = {Hadley Wickham and Jim Hester and Winston
Chang},
 year = {2019},
```

```
note = {R package version 2.0.2},
url = {https://CRAN.R-project.org/package=devtools},
}
```

6. The working directory is usually defined by the user by giving its path:

```
setwd("MY_WORKING_FOLDER_PATH")
```

The path can also be set within the *R* console tabs as follows: Misc > Change Working Directory. Then, the user can select the folder with the files to be analyzed. If the user wants to know the current working directory, he/she should type:

```
getwd()
```

7. The fasta (also known as FASTA/Pearson) format is a text representation of nucleotide or amino acid sequences. To reconstruct this type of format in *R*, the first line starts with the ">" symbol preceded by a summary text about the sequence such as its accession number, species origin, or other descriptive. The second line will include the actual sequence of nucleotides or amino acids. See examples below:

```
>Pan_troglodytes_NP_001502.1
MARAALSAAPSNPXXXXXXXXXXXXXXXXXXXXXXSVATELRCQCLQTLQGIHPKNIQSVN
VKSPGPHCAQTEVIATLKNGRKACLNPASPIVKKIIEKMLNSDKSN

>Homo_sapiens_chemokine_CXC_motif_ligand_1_BC011976.1
ATGGCCCGCGCTGCTCTCTCCGCCGCCCCCAGCAATCCCCGGCTCCTGCGAGTGGCACTGC
TGCTCCTGCTCCTGGTAGCCGCTGGCCGGCGCGCAGCAGGAGCGTCCGTGGCCACTGAACT
GCGCTGCCAGTGCTTGCAGACCCTGCAGGGAATTCACCCCAAGAACATCCAAAGTGTGAAC
GTGAAGTCCCCCGGACCCCACTGCGCCCAAACCGAAGTCATAGCCACACTCAAGAATGGGC
GGAAAGCTTGCCTCAATCCTGCATCCCCCATAGTTAAGAAAATCATCGAAAAGATGCTGAA
CAGTGACAAATCCAACT
```

8. Opening a console for programing in most platforms could be cumbersome. For OS X (Mac), this can be called by finding the "terminal" application in Applications > Utilities and clicking on the "terminal" application. For Windows, the user needs to call the command prompt. A simple approach is to create a small batch file using "NotePad" or any other text editor. This file should include the RAxML commands in one line and the "pause" command in the next line:

```
raxmlHPC-PTHREADS-SSE3.exe -T 10 -m GTRGAMMA -N 10 -s
CXCL_diversity_nucleotide_human_MUSCLE.fasta -p 12345 -n
CXC_nucleotide_raxml_trees
pause
```

This file is saved as a batch file (with a ".bat" file name extension) in the same folder with the RAxML executable (e.g., raxmlHPC-PTHREADS-SSE3.exe) and the input data file (CXCL_diversity_nucleotide_human_MUSCLE.fasta). To run the RAxML analysis on Windows, the user should double-click on the *.bat file.

9. The user can add the argument: -k which specifies that the bootstrapped trees should be saved with branch lengths. However, these analyses would take longer to run.

```
# for amino acids
./raxmlHPC-PTHREADS-SSE3 -T 10 -b 123476 -m PROTGAMMAAUTO -N
100 -s CXC_human_mouse_aminoacid_aligned_MUSCLE.fasta -p 12345
-n CXC_raxml_boot_trees

# for nucleotides
./raxmlHPC-PTHREADS-SSE3 -T 10 -b 123476 -m GTRGAMMA -N 100
-s CXCL_diversity_nucleotide_human_MUSCLE.fasta -p 12345 -k
-n CXC_nucleotide_raxml_boot_trees
```

10. A note from the *R*-package "phangorn" says: "Unfortunately the multicore package does not work under windows or with GUI interfaces ("aqua" on a mac)." The user should use this argument as "multicore = FALSE."
11. I noticed that some of the bootstrap values provided by the function "phangorn::bootstrap.pml" appear to be inflated (i.e., higher than expected) if compared with those obtained using RAxML. Therefore, I recommend using RAxML for bootstrap estimation.
12. GUI tree visualizing software: FigTree v1.4.3 (http://tree.bio.ed.ac.uk/software/figtree/); Mesquite v3.6 [30] (https://www.mesquiteproject.org/). More flexible tools for tree drawing are provided in the *R*-package "ggtree" [31].
13. Every *R*-package has a pdf manual usually found in CRAN, but the user can find the information for most functions by typing: ?NAME_FUNCTION within the *R* environment. This action will open a window where the user can read the main arguments required to run such function and, in most cases, a small example of this function's usage. However, the library that includes such function needs to be loaded first (e.g., library ("ape")) before we can read the manual.
14. The estimation of phylogenetic trees is a well-established field in comparative biology. The *R* environment is an ever-growing ecosystem, and many packages are published every day while others become obsolete or superseded by newer, faster, and more flexible ones. In spite of this, *R* keeps its spotlight by its

core accessibility to most biologists interested in sequence manipulation and phylogenetics. Like *R*-packages, phylogenetic programs also evolve toward faster, more flexible, and accommodating applications amenable for larger and more complex molecular data. The most successful programs are those that are kept current with the increasing computer capabilities in processing power. Likewise, most phylogenetic programs are adapting to genomic/proteomic levels of data that can only be handled with automatization. To keep up with this dynamism, users are encouraged to visit the following online resources and archives:

The CRAN Task View—Phylogenetics, Especially Comparative Methods: Given the size of CRAN (April 2019: 14086 available packages), this website provides a specialized collection of links to current *R*-packages related to phylogenetics and comparative methods.

https://cran.r-project.org/web/views/Phylogenetics.html

Bioconductor—This web-archive provides an easy access to tools and software for the analysis and comprehension of high-throughput genomic data. Bioconductor archives many *R*-packages not available in CRAN:

https://www.bioconductor.org/

References

1. Felsenstein J (2004) Inferring phylogenies. Sinauer Associates, Sunderland, MA
2. Gluckman P, Beedle A, Hanson M (2009) Principles of evolutionary medicine, 1st edn. Oxford University Press, Inc., New York
3. Garland T, Bennett AF, Rezende EL (2004) Phylogenetic approaches in comparative physiology. J Exp Biol 208:3015–3035
4. Baum D (2008) Reading a phylogenetic tree: the meaning of monophyletic groups. Nat Educ 1(1):190
5. Rosenberg M (2008) Sequence alignment: methods, models, concepts, and strategies. University of California Press, Berkeley, CA
6. R_Core_Team (2019) R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna. https://www.r-project.org/
7. Bachelerie F, Ben-Baruch A, Burkhardt AM, Combadiere C, Farber JM, Graham GJ, Horuk R, Sparre-Ulrich AH, Locati M, Luster AD, Mantovani A, Matsushima K, Murphy PM, Nibbs R, Nomiyama H, Power CA, Proudfoot AEI, Rosenkilde MM, Rot A, Sozzani S, Thelen M, Yoshie O, Zlotnik A (2014) International Union of Pharmacology. LXXXIX. Update on the extended family of chemokine receptors and introducing a new nomenclature for atypical chemokine receptors. Pharmacol Rev 66(1):1–79. https://doi.org/10.1124/pr.113.007724
8. Bateman A, Martin MJ, O'Donovan C, Magrane M, Apweiler R, Alpi E, Antunes R, Ar-Ganiska J, Bely B, Bingley M, Bonilla C, Britto R, Bursteinas B, Chavali G, Cibrian-Uhalte E, Da Silva A, De Giorgi M, Dogan T, Fazzini F, Gane P, Cas-Tro LG, Garmiri P, Hatton-Ellis E, Hieta R, Huntley R, Legge D, Liu WD, Luo J, MacDougall A, Mutowo P, Nightin-Gale A, Orchard S, Pichler K, Poggioli D, Pundir S, Pureza L, Qi GY, Rosanoff S, Saidi R, Sawford T, Shypitsyna A, Turner E, Volynkin V, Wardell T, Watkins X, Watkins CA, Figueira L, Li WZ, McWilliam H, Lopez R, Xenarios I, Bougueleret L, Bridge A, Poux S, Redaschi N, Aimo L, Argoud-Puy G, Auchincloss A, Axelsen K, Bansal P, Baratin D, Blatter MC, Boeckmann B, Bolleman J, Boutet E, Breuza L, Casal-Casas C, De Castro E, Coudert E, Cuche B, Doche M, Dornevil D, Duvaud S, Estreicher A, Famiglietti L, Feuermann M, Gasteiger E, Gehant S, Gerritsen V, Gos A, Gruaz-Gumowski N, Hinz U, Hulo C, Jungo F, Keller G, Lara V, Lemercier P, Lieberherr D,

Lombardot T, Martin X, Masson P, Morgat A, Neto T, Nouspikel N, Paesano S, Pedruzzi I, Pilbout S, Pozzato M, Pruess M, Rivoire C, Roechert B, Schneider M, Sigrist C, Sonesson K, Staehli S, Stutz A, Sundaram S, Tognolli M, Verbregue L, Veuthey AL, Wu CH, Arighi CN, Arminski L, Chen CM, Chen YX, Garavelli JS, Huang HZ, Laiho KT, McGarvey P, Natale DA, Suzek BE, Vinayaka CR, Wang QH, Wang YQ, Yeh LS, Yerramalla MS, Zhang J, UniProt C (2015) UniProt: a hub for protein information. Nucleic Acids Res 43(D1):D204–D212. https://doi.org/10.1093/nar/gku989

9. Wickham H, Hester J, Chang W (2019) devtools: tools to make developing R packages easier. R package version 2.0.2. https://cran.r-project.org/package=devtools
10. Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, Bravo HC, Davis S, Gatto L, Girke T, Gottardo R, Hahne F, Hansen KD, Irizarry RA, Lawrence M, Love MI, MacDonald J, Obenchain V, Oles AK, Pages H, Reyes A, Shannon P, Smyth GK, Tenenbaum D, Waldron L, Morgan M (2015) Orchestrating high-throughput genomic analysis with bioconductor. Nat Methods 12 (2):115–121. https://doi.org/10.1038/nmeth.3252
11. Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge YC, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JYH, Zhang JH (2004) Bioconductor: open software development for computational biology and bioinformatics. Genome Biol 5 (10):16. https://doi.org/10.1186/gb-2004-5-10-r80
12. Gentleman R (2019) annotate: annotation for microarrays. R package version 1.60.1
13. Paradis E, Schliep K (2018) ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. Bioinformatics 35:526–528
14. Paradis E (2012) Analysis of phylogenetics and evolution with R, 2nd edn. Springer, New York
15. Schliep KP (2011) phangorn: phylogenetic analysis in R. Bioinformatics 27(4):592–593
16. Winter DJ (2017) rentrez: an R package for the NCBI eUtils API. The R Journal 9 (2):520–526
17. Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, Bealer K, Madden TL (2009) BLAST plus: architecture and applications. BMC Bioinformatics 10:9. https://doi.org/10.1186/1471-2105-10-421
18. Sievers F, Wilm A, Dineen D, Gibson TJ, Karplus K, Li WZ, Lopez R, McWilliam H, Remmert M, Soding J, Thompson JD, Higgins DG (2011) Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Mol Syst Biol 7:6. https://doi.org/10.1038/msb.2011.75
19. Goujon M, McWilliam H, Li WZ, Valentin F, Squizzato S, Paern J, Lopez R (2010) A new bioinformatics analysis tools framework at EMBL-EBI. Nucleic Acids Res 38: W695–W699. https://doi.org/10.1093/nar/gkq313
20. Gouy M, Guindon S, Gascuel O (2010) SeaView version 4: a multiplatform graphical user interface for sequence alignment and phylogenetic tree building. Mol Biol Evol 27 (2):221–224. https://doi.org/10.1093/molbev/msp259
21. Edgar RC (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res 32 (5):1792–1797. https://doi.org/10.1093/nar/gkh340
22. Stamatakis A (2014) RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. Bioinformatics 30 (9):1312–1313. https://doi.org/10.1093/bioinformatics/btu033
23. Darriba D, Taboada GL, Doallo R, Posada D (2012) jModelTest 2: more models, new heuristics and parallel computing. Nat Methods 9 (8):772–772. https://doi.org/10.1038/nmeth.2109
24. Guindon S, Dufayard JF, Lefort V, Anisimova M, Hordijk W, Gascuel O (2010) New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0. Syst Biol 59 (3):307–321. https://doi.org/10.1093/sysbio/syq010
25. Tamura K, Nei M (1993) Estimation of the number of nucleotide substitutions in the control region of mitochondrial-DNA in humans and chimpanzees. Mol Biol Evol 10 (3):512–526
26. Jones DT, Taylor WR, Thornton JM (1992) The rapid generation of mutation data matrices from protein sequences. Comput Appl Biosci 8 (3):275–282
27. Jukes TH, Cantor CR (1969) Evolution of protein molecules. In: Munro HN (ed) Mammalian protein metabolism. Academic Press, New York, pp 21–132
28. Felsenstein J (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. J Mol Evol 17(6):368–376

29. Saitou N, Nei M (1987) The neighbor-joining method—a new method for reconstructing phylogenetic trees. Mol Biol Evol 4 (4):406–425
30. Maddison WP, Maddison DR (2018) Mesquite: a modular system for evolutionary analysis. Version 3.51. http://www.mesquiteproject.org/
31. Yu GC, Smith DK, Zhu HC, Guan Y, Lam TTY (2017) GGTREE: an R package for visualization and annotation of phylogenetic trees with their covariates and other associated data. Methods Ecol Evol 8(1):28–36. https://doi.org/10.1111/2041-210x.12628