Look for co-evolution

Elena Montenegro, Sara González, Sofía Cerdá, Fernando Freire

January 4, 2019

Contents

1 Look for coevolution Streptococcus-Lactobacillus

Coevolution is everywhere. Because no species can be considered totally isolated from the others. And not only on a biological level, we perceive it in the lines of change in our societies, in political, scientific, religious ideas and even in the evolution of software engineering.

The difficulty is to know how much co-evolution is due to the interaction between a pair of species, because the tandem can not be totally isolated from the rest of the universe either. No doubt the problem can not be solved only from the bioinformatic perspective, it is also necessary the contribution of other branches of knowledge such as microbiology, physics, mathematics and computer theory.

Our approach consists in comparing the rates of protein evolutionary change between strains of Streptococcus and Lactobacillus linked by symbiotic pathways, and the rates between the rest of species of each genus.

We postulate that the proteins that reflects a rate change with more significance, i.e., more speedy or more slowly that intra genus rates, are proteins that could be influenced by the symbiotic environment.

To do so we need to compute separately both species of symbiotic tandem. The detailed steps are:

1) Obtain all the proteome from this four groups of species:

- Genus streptococcus
- Strains of streptococcus termophilus.
- Genus lactobacillus
- Strains of lactobacillus bulgaricus.

2) For each protein of each group we obtain the phylogenetic tree and compute the mean branch length. We suppose that we have ultrametricicy or almost ultrametricity. It can be not exact but it could serve as a reference. We use the clulstalw multialignment method, but other methods as T-COFFEE or MUSCLE could be used too. The branch length is a measure of the substitution rate.

3) For each protein of the first an second group, we calculate the ration between branch length of termophilus and branch length of protein in genus. And also we compute the overall mean of all ratios. Finally we select the proteins with the most extreme values (80% percentile, two tailed).

4) We do the same with lactobacillus (third an fourth groups). At this stage we have two sets of proteins.

5) We obtain the biologic pathways of each of the sets of proteins.

6) The proteins involved in the same pathways from lactobacillus ans streptoccocus are the target proteins influenced by the coevolution between the two species. It will be necessary a later microbiological study that confirms this expectancies and dive deeply in the details of this coevolution.

1.1 Obtain data from servers

1.1.1 Methods

Script 1.1.1 (python)

```python
import requests, sys
import pickle

# Disc serialization methods
def dump(obj, obj_name):
```

```
    binary_file = open(obj_name + '.bin',mode='wb')
    pickled_obj = pickle.dump(obj, binary_file)
    binary_file.close()

def load(obj_name):
    with (open(obj_name + ".bin", "rb")) as openfile:
        while True:
            try:
                obj = pickle.load(openfile)
            except EOFError:
                break
    return obj

def load_taxa(scientific_prefix):
    """
    """
    requestURL = "https://www.ebi.ac.uk/proteins/api/taxonomy/name/" + scientific_prefix +\
                 "%20?pageNumber=1&pageSize=100&searchType=STARTSWITH&fieldName=SCIENTIFICNAM
E"

    r = requests.get(requestURL, headers={ "Accept" : "application/json"})

    if not r.ok:
      r.raise_for_status()
      sys.exit()

    jsonBody = json.loads(r.text)
    taxa = []
    names = []
    for taxonomy in jsonBody["taxonomies"]:
        print(taxonomy['taxonomyId'], taxonomy['scientificName'])
        taxa.append(taxonomy['taxonomyId'])
        names.append(taxonomy['scientificName'])
    return taxa, names

def load_proteome(taxids, size=10, protein=["LDH"]):
    """
    """
    taxids_str = ",".join(str(x) for x in taxids)
    protein_str = ",".join(x for x in protein)
    print(taxids_str)
    requestURL = "https://www.ebi.ac.uk/proteins/api/proteins?offset=0&size=" + str(size) +
    ↪  "&taxid=" +\
                    taxids_str + "&reviewed=false"
    if protein != []:
        requestURL += "&gene=" + protein_str
    print(requestURL)
    r = requests.get(requestURL, headers={ "Accept" : "text/x-fasta"})

    if not r.ok:
      r.raise_for_status()
      sys.exit()
```

```
56
57      proteome = r.text
58      return proteome
```

1.1.2 Load streptococcus taxids

Script 1.1.2 (python)

```
1 termophilus_taxa,  termophilus_names = load_taxa("Streptococcus thermophilus")
2 streptococcus_taxa,  streptococcus_names = load_taxa("Streptococcus")
3 dump(termophilus_taxa, "termophilus_taxa")
4 dump(termophilus_names, "termophilus_names")
5 dump(streptococcus_taxa, "streptococcus_taxa")
6 dump(streptococcus_names, "streptococcus_names")
```

Output

```
264199 Streptococcus thermophilus (strain ATCC BAA-250 / LMG 18311)
299768 Streptococcus thermophilus (strain CNRZ 1066)
322159 Streptococcus thermophilus (strain ATCC BAA-491 / LMD-9)
767463 Streptococcus thermophilus (strain ND03)
1042404 Streptococcus thermophilus CNCM I-1630
1051074 Streptococcus thermophilus JIM 8232
1073569 Streptococcus thermophilus MTCC 5460
1073570 Streptococcus thermophilus MTCC 5461
1091038 Streptococcus thermophilus DSM 20617
1187956 Streptococcus thermophilus MN-ZLW-002
1263110 Streptococcus thermophilus CAG:236
1268061 Streptococcus thermophilus DGCC 7710
1408178 Streptococcus thermophilus ASCC 1275
1415776 Streptococcus thermophilus TH1435
1423145 Streptococcus thermophilus TH1436
1433288 Streptococcus thermophilus MTH17CL396
1433289 Streptococcus thermophilus M17PTZA496
1435972 Streptococcus thermophilus TH985
1435974 Streptococcus thermophilus TH982
1435981 Streptococcus thermophilus 1F8CT
1436725 Streptococcus thermophilus TH1477
1302 Streptococcus gordonii
1303 Streptococcus oralis
1304 Streptococcus salivarius
1305 Streptococcus sanguinis
1306 Streptococcus sp.
1307 Streptococcus suis
1308 Streptococcus thermophilus
1309 Streptococcus mutans
1310 Streptococcus sobrinus
1311 Streptococcus agalactiae
1313 Streptococcus pneumoniae
1314 Streptococcus pyogenes
```

```
1317 Streptococcus downei
1318 Streptococcus parasanguinis
1319 Streptococcus sp. 'group B'
1320 Streptococcus sp. group G
1324 Streptococcus sp. G148
1325 Streptococcus sp. GX7805
1326 Streptococcus acidominimus
1328 Streptococcus anginosus
1329 Streptococcus canis
1332 Streptococcus criae
1333 Streptococcus criceti
1334 Streptococcus dysgalactiae
1335 Streptococcus equinus
1336 Streptococcus equi
1337 Streptococcus hyointestinalis
1338 Streptococcus intermedius
1339 Streptococcus macacae
1340 Streptococcus porcinus
1341 Streptococcus ratti
1343 Streptococcus vestibularis
1345 Streptococcus ferus
1346 Streptococcus iniae
1348 Streptococcus parauberis
1349 Streptococcus uberis
10728 Streptococcus pneumoniae phage HB-3
10747 Streptococcus phage Cp-1
10748 Streptococcus phage Cp-7
10749 Streptococcus phage Cp-9
12344 Streptococcus phage Cp-5
12366 Streptococcus pyogenes phage H4489A
12402 Streptococcus phage EJ-1
28037 Streptococcus mitis
29389 Streptococcus alactolyticus
29390 Streptococcus gordonii str. Challis
33040 Streptococcus milleri
33972 Streptococcus sp. 'group C'
35344 Streptococcus pyogenes phage T12
36470 Streptococcus sp. 'group A'
39425 Streptococcus phage T270
40041 Streptococcus equi subsp. zooepidemicus
40287 Streptococcus mutans serotype C
45634 Streptococcus cristatus
53354 Streptococcus gallolyticus
55085 Streptococcus thoraltensis
59310 Streptococcus macedonicus
64186 Streptococcus virus Sfi21
68891 Streptococcus peroris
68892 Streptococcus infantis
69017 Streptococcus sp. (strain 19909)
72638 Streptococcus virus Sfi19
73422 Streptococcus phage TP-J34
73492 Streptococcus pyogenes phage
```

```
74382 Streptococcus phage SFi18
76860 Streptococcus constellatus
78535 Streptococcus viridans
78541 Streptococcus virus Sfi11
82269 Streptococcus sp. 28D
82348 Streptococcus pluranimalium
82806 Streptococcus ovis
83545 Streptococcus sp. KN1
83546 Streptococcus sp. KN2
83547 Streptococcus sp. KN3
83549 Streptococcus sp. TW1
85154 Streptococcus phage O1205
86065 Streptococcus pyogenes phage H10403
90410 Streptococcus virus DT1
99822 Streptococcus dysgalactiae subsp. dysgalactiae
102143 Streptococcus phage S3b
102144 Streptococcus phage S92
102145 Streptococcus phage ST3
102146 Streptococcus phage ST64
102147 Streptococcus phage Sfi16A
102150 Streptococcus phage J1
102684 Streptococcus infantarius
102886 Streptococcus didelphis
104215 Streptococcus sp. 1400-98
112023 Streptococcus virus 7201
113107 Streptococcus australis
114652 Streptococcus orisratti
116154 Streptococcus sp. Z1227
116155 Streptococcus sp. Z12
116156 Streptococcus sp. Z89
118670 Streptococcus sp. PSH2
118671 Streptococcus sp. PSH1a
118672 Streptococcus sp. PSH1b
119224 Streptococcus phocae
119602 Streptococcus dysgalactiae subsp. equisimilis
119603 Streptococcus dysgalactiae group
```

1.1.3 Load lactobacillus taxids

Script 1.1.3 (python)

```python
lactobacillus_taxa,  lactobacillus_names = load_taxa("Lactobacillus")
bulgaricus_taxa,  bulgaricus_names = load_taxa("Lactobacillus delbrueckii subsp. bulgaricus")

dump(lactobacillus_taxa, "lactobacillus_taxa")
dump(lactobacillus_names, "lactobacillus_names")
dump(bulgaricus_taxa, "bulgaricus_taxa")
dump(bulgaricus_names, "bulgaricus_names")
```

Output

```
1579 Lactobacillus acidophilus
1580 Lactobacillus brevis
1581 Lactobacillus buchneri
1582 Lactobacillus casei
1584 Lactobacillus delbrueckii
1585 Lactobacillus delbrueckii subsp. bulgaricus
1587 Lactobacillus helveticus
1588 Lactobacillus hilgardii
1589 Lactobacillus pentosus
1590 Lactobacillus plantarum
1591 Lactobacillus sp.
1593 Lactobacillus sp. (strain 30a)
1596 Lactobacillus gasseri
1597 Lactobacillus paracasei
1598 Lactobacillus reuteri
1599 Lactobacillus sakei
1600 Lactobacillus acetotolerans
1601 Lactobacillus agilis
1602 Lactobacillus alimentarius
1603 Lactobacillus amylophilus
1604 Lactobacillus amylovorus
1605 Lactobacillus animalis
1606 Lactobacillus aviarius
1607 Lactobacillus bifermentans
1610 Lactobacillus coryniformis
1612 Lactobacillus farciminis
1613 Lactobacillus fermentum
1614 Lactobacillus fructivorans
1618 Lactobacillus mali
1622 Lactobacillus murinus
1623 Lactobacillus ruminis
1624 Lactobacillus salivarius
1625 Lactobacillus sanfranciscensis
1626 Lactobacillus sharpeae
1627 Lactobacillus thermophilus
1628 Lactobacillus vermiforme
1632 Lactobacillus oris
1633 Lactobacillus vaginalis
1634 Lactobacillus sakei L45
12417 Lactobacillus phage phiadh
28038 Lactobacillus curvatus
28039 Lactobacillus leichmannii
29273 Lactobacillus phage J1
29397 Lactobacillus delbrueckii subsp. lactis
29398 Lactobacillus heterohiochii
29399 Lactobacillus japonicus
33959 Lactobacillus johnsonii
33960 Lactobacillus collinoides
33961 Lactobacillus homohiochii
33962 Lactobacillus kefiri
35787 Lactobacillus pontis
```

```
37105 Lactobacillus phage JCL1032
39103 Lactobacillus phage PL-1
44272 Lactobacillus helveticus subsp. jugurti
47493 Lactobacillus panis
47714 Lactobacillus paracasei subsp. paracasei
47715 Lactobacillus rhamnosus
47770 Lactobacillus crispatus
51369 Lactobacillus phage A2
51664 Lactobacillus dextrinicus
52242 Lactobacillus gallinarum
52979 Lactobacillus phage phig1e
53444 Lactobacillus lindneri
57037 Lactobacillus zeae
60519 Lactobacillus graminis
60520 Lactobacillus paraplantarum
81857 Lactobacillus selangorensis
82688 Lactobacillus nagelii
83526 Lactobacillus paralimentarius
83683 Lactobacillus amylolyticus
83684 Lactobacillus delbrueckii subsp. delbrueckii
88164 Lactobacillus fornicalis
88233 Lactobacillus manihotivorans
88430 Lactobacillus sp. LM-17
89059 Lactobacillus acidipiscis
89060 Lactobacillus sp. FS1111
91019 Lactobacillus phage phiFSW
94706 Lactobacillus sp. MRS-III06
94707 Lactobacillus sp. MRS-II22
96565 Lactobacillus hamsteri
97137 Lactobacillus sp. ASF360
97478 Lactobacillus mucosae
100468 Lactobacillus perolens
104955 Lactobacillus frumenti
105612 Lactobacillus algidus
106493 Lactobacillus phage LL-K
106576 Lactobacillus sp. C33LV5
106577 Lactobacillus sp. G24
109790 Lactobacillus jensenii
113051 Lactobacillus sp. GTH2
113052 Lactobacillus sp. GTH24
113053 Lactobacillus sp. GTH28
113054 Lactobacillus sp. GTH6
113055 Lactobacillus sp. GTH26
113056 Lactobacillus sp. GTP5
113057 Lactobacillus sp. GTS2
113058 Lactobacillus sp. GTH22
113059 Lactobacillus sp. GTH18
113060 Lactobacillus sp. GTH29
113061 Lactobacillus sp. JN1
321956 Lactobacillus delbrueckii subsp. bulgaricus (strain ATCC BAA-365)
353496 Lactobacillus delbrueckii subsp. bulgaricus (strain 2038)
```

```
390333 Lactobacillus delbrueckii subsp. bulgaricus (strain ATCC 11842 / DSM 20081 / JCM 1002
↪  / NBRC 13953 / NCIMB 11778)
767455 Lactobacillus delbrueckii subsp. bulgaricus (strain ND02)
784613 Lactobacillus delbrueckii subsp. bulgaricus PB2003/044-T3-4
1042399 Lactobacillus delbrueckii subsp. bulgaricus CNCM I-1632
1042400 Lactobacillus delbrueckii subsp. bulgaricus CNCM I-1519
```

1.1.4 Load streptococcus proteomes

Script 1.1.4 (python)

```python
RESTART = True
VERBOSE = True
# If [] return all the proteins
PROTEIN_LIST = []

if RESTART:
    termophilus_taxa = load("termophilus_taxa")
    streptococcus_taxa = load("streptococcus_taxa")

# We limit to 20 taxids (it's the maximum for webservice and it should be enough
termophilus_taxids = termophilus_taxa[0:19]
streptococcus_taxids = streptococcus_taxa[0:19]
print(streptococcus_taxids)
print(termophilus_taxids)

#streptococcus_proteome = load_proteome(streptococcus_taxids, -1, protein = ["LDH", "CAS2",
↪  "CAS3"])
#termophilus_proteome = load_proteome(termophilus_taxids, -1, protein = ["LDH", "CAS2",
↪  "CAS3"])

streptococcus_proteome_complete = load_proteome(streptococcus_taxids, -1, PROTEIN_LIST)
termophilus_proteome_complete = load_proteome(termophilus_taxids, -1, PROTEIN_LIST)

dump(streptococcus_proteome_complete, "streptococcus_proteome_complete")
dump(termophilus_proteome_complete, "termophilus_proteome_complete")
```

Output

```
[1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1313, 1314, 1317, 1318, 1319,
↪  1320, 1324, 1325, 1326]
[264199, 299768, 322159, 767463, 1042404, 1051074, 1073569, 1073570, 1091038, 1187956,
↪  1263110, 1268061, 1408178, 1415776, 1423145, 1433288, 1433289, 1435972, 1435974]
1302,1303,1304,1305,1306,1307,1308,1309,1310,1311,1313,1314,1317,1318,1319,1320,1324,1325,1326
https://www.ebi.ac.uk/proteins/api/proteins?offset=0&size=-1&taxid=1302,1303,1304,1305,1306,1
↪  307,1308,1309,1310,1311,1313,1314,1317,1318,1319,1320,1324,1325,1326&reviewed=false
264199,299768,322159,767463,1042404,1051074,1073569,1073570,1091038,1187956,1263110,1268061,1
↪  408178,1415776,1423145,1433288,1433289,1435972,1435974
```

```
https://www.ebi.ac.uk/proteins/api/proteins?offset=0&size=-1&taxid=264199,299768,322159,76746
↪ 3,1042404,1051074,1073569,1073570,1091038,1187956,1263110,1268061,1408178,1415776,1423145
↪ ,1433288,1433289,1435972,1435974&reviewed=false
```

Script 1.1.5 (python)

```python
print(termophilus_proteome_complete[0:1000])
print(streptococcus_proteome_complete[0:1000])
```

Output

```
>tr|V6CG59|V6CG59_STRTN Proteolysis tag peptide encoded by tmRNA Strep_therm_CNRZ10
↪ (Fragment) OS=Streptococcus thermophilus (strain ND03) OX=767463 GN=tmRNA
↪ Strep_therm_CNRZ10 PE=4 SV=1
AKNTNSYAVAA
>tr|V6CG54|V6CG54_STRTR Proteolysis tag peptide encoded by tmRNA Strep_therm (Fragment)
↪ OS=Streptococcus thermophilus MN-ZLW-002 OX=1187956 GN=tmRNA Strep_therm PE=4 SV=1
AKNTNSYAVAA
>tr|V6CG50|V6CG50_STRTR Proteolysis tag peptide encoded by tmRNA Strep_therm_CNRZ10
↪ (Fragment) OS=Streptococcus thermophilus MTCC 5461 OX=1073570 GN=tmRNA Strep_therm_CNRZ10
↪ PE=4 SV=1
AKNTNSYAVAA
>tr|V6BJW4|V6BJW4_STRTR Proteolysis tag peptide encoded by tmRNA Strep_therm_CNRZ10
↪ (Fragment) OS=Streptococcus thermophilus MTCC 5460 OX=1073569 GN=tmRNA Strep_therm_CNRZ10
↪ PE=4 SV=1
AKNTNSYAVAA
>tr|V6CE29|V6CE29_STRT1 Proteolysis tag peptide encoded by tmRNA Strep_therm_CNRZ10
↪ (Fragment) OS=Streptococcus thermophilus (strain CNRZ 1066) OX=299768 GN=tmRNA
↪ Strep_therm_CNRZ10 PE=4 SV=1
AKNTNSYAVAA
>tr|A0A1L1QK15|A0A1L1Q
>tr|D1MQ05|D1MQ05_STRDO Sortase OS=Streptococcus downei OX=1317 GN=srtA PE=4 SV=1
MKRTSSRNSKGRKEKESSKKKHHWLRTILVLFLMVVGLALVFNRSIRNSVIAWNTNKYQV
SKVDKKTLTKNKKAKVNYDFDSVKSISAQSVISSQMDAQDLPVIGGIAIPDLELNLPIFK
GLGNTELSYGAGTMKESQVMGGENNYALASHHVFGVNGASKMLFSPLENAKNGMEIYLTD
KNKVYTYIISEVKTVEPSDVAVINDTPGAKNLTLVTCDDAEATNRIIVSANYKEEVSYDK
ASQKMIEAFNRSYNQLSL
>tr|A7VLR2|A7VLR2_STRDO Autolysin OS=Streptococcus downei OX=1317 GN=atlh PE=4 SV=1
MRKKMYKVKKHWVIASAAALTVLGTSGLVAADEQSYPQTGASRELASVSTDPSVTATDTQ
DSPKSVENTSQPASTSTLDDSSQEGQNQLVDETSQSSQSEPAGSETISDWETDNQSASEV
QSSQRDSEAQIQAPASTNQVDPSDEEDQLQVADQTINEGSRVLRASAGATVARATAAGQS
SSEVTSASVSGKTLIIHYNRSIASNEALYFAVWTEENDQDDMVWYQANAQGAAYIDLSKH
RFYGKYIIHTYSVINGQNIGRNATSIMVNPPKISSKIEATSNGDYKLTVGNVSSDITEVF
VPVWSDKNDQDDMIWYHADKVNANTYSLTIKPANHHNDFDHYSVHIYGQSSITRGLIGLS
ATPGFERKETASSSKVAPQNKVRASLGANGIQLHLDTNEAADLSNIYFAVWSQVGDQDDI
HWYPADGSLSALAPYSNHSGYGTYHIHTYANKNSKFVGINTTSIEVPSPSASAKISKQDD
ATYNVKIDKVSSYITSVLVPVWTQKNDQDDIVWYPASKQSDGSYSATIKLKDHNFESGKY
LVHLYGYSSLEGGRLVGLGATD
```

1.1.5 Load lactobacillus proteomes

Script 1.1.6 (python)

```python
RESTART = True
VERBOSE = True
# If [] return all the proteins
PROTEIN_LIST = []

if RESTART:
    lactobacillus_taxa = load("lactobacillus_taxa")
    lactobacillus_names = load("lactobacillus_names")
    bulgaricus_taxa = load("bulgaricus_taxa")
    bulgaricus_names = load("bulgaricus_names")
# We limit to 20 taxids (it's the maximum for webservice and it should be enough
lactobacillus_taxids = lactobacillus_taxa[0:19]
bulgaricus_taxids = bulgaricus_taxa[0:19]
print(lactobacillus_taxids)
print(bulgaricus_taxids)

lactobacillus_proteome_complete = load_proteome(lactobacillus_taxids, -1, PROTEIN_LIST)
bulgaricus_proteome_complete = load_proteome(bulgaricus_taxids, -1, PROTEIN_LIST)

dump(lactobacillus_proteome_complete, "lactobacillus_proteome_complete")
dump(bulgaricus_proteome_complete, "bulgaricus_proteome_complete")
```

Output

```
[1579, 1580, 1581, 1582, 1584, 1585, 1587, 1588, 1589, 1590, 1591, 1593, 1596, 1597, 1598,
↪  1599, 1600, 1601, 1602]
[321956, 353496, 390333, 767455, 784613, 1042399, 1042400]
1579,1580,1581,1582,1584,1585,1587,1588,1589,1590,1591,1593,1596,1597,1598,1599,1600,1601,1602
https://www.ebi.ac.uk/proteins/api/proteins?offset=0&size=-1&taxid=1579,1580,1581,1582,1584,1
↪  585,1587,1588,1589,1590,1591,1593,1596,1597,1598,1599,1600,1601,1602&reviewed=false
321956,353496,390333,767455,784613,1042399,1042400
https://www.ebi.ac.uk/proteins/api/proteins?offset=0&size=-1&taxid=321956,353496,390333,76745
↪  5,784613,1042399,1042400&reviewed=false
```

1.2 Compute substitution rates

1.2.1 Methods

Script 1.2.1 (python)

```python
import re

def proteome2dict(proteome_fasta):
    """
    Returns a dict with keys protein accession and values the list of fasta format for all
↪   taxids
```

```
    This is the basis for clustalw alignments and tree generation
    """
    proteome = {}
    key_found = False
    for line in proteome_fasta.splitlines():
        if len(line) > 0:
            if line[0] == ">":
                if key_found:
                    if key in proteome:
                        proteome[key].append(seq)
                    else:
                        proteome[key] = [seq]
                search_gene_name = re.search('GN=(\w*)', line)
                if search_gene_name:
                    key = search_gene_name.group(1).upper()
                    key_found = True
                #print(key)
                seq = line + '\n'
            elif key_found:
                seq += line + '\n'
    if key_found:
        if key in proteome:
            proteome[key].append(seq)
        else:
            proteome[key] = [seq]
    return proteome
```

Script 1.2.2 (python)

```
# Phylo tree with clustalw. We need to measure the substitution rate.
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
from Bio import Phylo
from io import StringIO
import os
from Bio.Align.Applications import ClustalwCommandline
CLUSTALW = r"./clustalw2"
assert os.path.isfile(CLUSTALW), "Clustal W executable missing"
plt.rcParams["figure.figsize"] = (20,30)
matplotlib.rc('font', size=12)

def compute_mean_subst_rate(proteome, verbose=False, show_tree=False):
    """
    """
    clustalw_cline = ClustalwCommandline(CLUSTALW, infile=proteome + ".fasta")
    stdout, stderr = clustalw_cline()
    f = open(proteome + ".dnd", "r")
    s_tree = f.read()
    f.close()
    #print(s_tree)
```

```
    branch_len = 0
    num_branches = 0
    search_branch_length = re.findall(':([-.0123456789]*)', s_tree)
    for branch_length in search_branch_length:
        #print(branch_length)
        if branch_length != "0.00000":
            branch_len += float(branch_length)
            num_branches += 1
    if num_branches > 0:
        rate = branch_len/num_branches
    else:
        rate = -1
    if verbose: print(branch_len, num_branches, rate)
    if show_tree:
        tree = Phylo.read(proteome + ".dnd", "newick")
        Phylo.draw(tree)
    return rate

def compute_subst_rates(proteome, proteome_keys, proteome_name, verbose=False):
    """
    """
    subst_rates = {}
    for protein in proteome_keys:
        if verbose: print(protein)
        protein_sequence = ""
        # Only for proteins with enough sequences to make a tree
        if len(proteome[protein]) >= 3:
            for sequence in proteome[protein]:
                protein_sequence += sequence
            fasta_file_name = proteome_name + "_" + protein
            f = open(fasta_file_name + ".fasta", "w")
            if verbose: print(protein_sequence)
            f.write(protein_sequence)
            f.close()
            mean_subst_rate = compute_mean_subst_rate(fasta_file_name)
            subst_rates[protein] = mean_subst_rate
    return subst_rates
```

1.2.2 Fasta to dictionary

Script 1.2.3 (python)

```
#Proteomes in fasta to dictionaries
RESTART = True
VERBOSE = True

if RESTART:
    termophilus_proteome_fasta = load("termophilus_proteome_complete")
    streptococcus_proteome_fasta = load("streptococcus_proteome_complete")

```

```
#print(termophilus_proteome_fasta)
proteome_termophilus = proteome2dict(termophilus_proteome_fasta)
proteome_streptococcus = proteome2dict(streptococcus_proteome_fasta)

dump(proteome_termophilus, "proteome_termophilus")
dump(proteome_streptococcus, "proteome_streptococcus")

if VERBOSE: print(list(proteome_termophilus.keys())[0:100])
if VERBOSE: print(list(proteome_streptococcus.keys())[0:100])
```

Output

```
['TMRNA', 'BN551_00358', 'GALT', 'LACZ', 'SBCD', 'GALR', 'DNAN', 'STU0007', 'FTSH', 'MREC',
↪  'ARAT', 'PURL', 'STU0044', 'STU0052', 'STU0053', 'TAG', 'STU0075', 'STU0082', 'LABC',
↪  'STU0110', 'STU0113', 'ILVA', 'STU0161', 'STU0182', 'STU0202', 'STU0208', 'STU0251',
↪  'STU0258', 'STU0267', 'CBIM', 'STU0297', 'TRKH2', 'DNAB', 'SERS', 'STU0330', 'STU0334',
↪  'STU0338', 'STU0339', 'METB1', 'STU0358', 'LIVM', 'HPF', 'FABF', 'FRUR', 'STU0422',
↪  'STU0435', 'PHOH', 'STU0448', 'METS', 'STU0452', 'PRTM', 'ARGB', 'STU0468', 'STU0473',
↪  'STU0474', 'STU0475', 'FTSW', 'DNAH', 'NAGA', 'STU0510', 'STU0516', 'STU0539', 'STU0551',
↪  'STU0557', 'STU0565', 'HLYIII', 'STU0580', 'STU0595', 'MURN', 'NNRD', 'RNR', 'STU0631',
↪  'STU0636', 'AROE', 'MIP', 'CAS2', 'STU0665', 'STU0668', 'STU0672', 'STU0675', 'STU0678',
↪  'STU0679', 'STU0681', 'STU0693', 'STU0695', 'GPMC', 'STU0702', 'STU0704', 'LEMA',
↪  'STU0721', 'APBE', 'DLTX', 'STU0811', 'STU0819', 'STU0829', 'ADCA', 'MUR2', 'STU0876',
↪  'STU0877', 'STHIM']
['SRTA', 'ATLH', 'DBLB', 'RPOA', 'PHES', 'DEX', 'RPOB', 'GBPC3', 'GROEL', 'GTF', 'GROS',
↪  'SODA', 'DBLA', 'GYRB', 'RECA', 'ATPA', 'ERMTR', 'MEF', 'EMM', 'TUF', 'PAH', 'GBPC2',
↪  'GBPC1', 'ERMB', 'GFBA', 'SILB', 'SILCR', 'NADE', 'ENO1', 'SPAA', 'PROTEIN', 'EMML',
↪  'BLPM', 'SILD', 'SILC', 'SILA', 'SSO1', 'GTFU', 'GLUCOSYLTRANSFERASE', 'DEI', 'PAR',
↪  'CAS1', 'SILE', 'SU1', 'SU2', 'GTFI', 'GROES', 'DBLC', 'CPN60', 'CAS2', 'FTSK', 'GTFS',
↪  'CMK', 'GALU', 'SSO2', 'LCD', 'GTFT', 'PAG', 'FOLP', 'RODA', 'HISC', 'ATLG', 'GTFC',
↪  'DRPA', 'MUTF', 'SRLB', 'SRLE', 'SRLM', 'MUTC', 'ORF', 'SLOB', 'MUTA', 'MUTP', 'MUTG',
↪  'SATC', 'MUTR', 'COMC', 'RMPB', 'PSAA', 'MEPA', 'MBRA', 'RGPE', 'RGPA', 'RGPF', 'SMBG',
↪  'SMBB', 'GBPC', 'A6J86_007565', 'LPP', 'DDL', 'DLTD', 'ORF2', 'RGPBC', 'GTFB', 'GTFD',
↪  'PYK', 'SPAP', 'PPAC', 'LEPC', 'ATLA']
```

Script 1.2.4 (python)

```
#Proteomes in fasta to dictionaries
RESTART = True
VERBOSE = True

if RESTART:
    bulgaricus_proteome_fasta = load("bulgaricus_proteome_complete")
    lactobacillus_proteome_fasta = load("lactobacillus_proteome_complete")

proteome_bulgaricus = proteome2dict(bulgaricus_proteome_fasta)
proteome_lactobacillus = proteome2dict(lactobacillus_proteome_fasta)

dump(proteome_bulgaricus, "proteome_bulgaricus")
dump(proteome_lactobacillus, "proteome_lactobacillus")

```

```python
if VERBOSE: print(list(proteome_bulgaricus.keys())[0:100])
if VERBOSE: print(list(proteome_lactobacillus.keys())[0:100])
```

Output

```
['TMRNA', 'EXOA', 'LDB0010', 'LDB0021', 'LDB0026', 'LDB0031', 'LDB0191', 'LDB0200', 'PHNB',
 ↪  'LDB0206', 'LDB0207', 'NRDG', 'LDB0226', 'LDB0229', 'LDB0233', 'LDB0252', 'LDB0271',
 ↪  'OPPA3II', 'OPPCII', 'OPPDII', 'LDB0301', 'LDB0313', 'LDB0314', 'LDB0327', 'LDB0329',
 ↪  'LDB0332', 'LDB0351', 'CSHA', 'LDB0365', 'FTSH', 'LYSS', 'LDB0374', 'LDB0378', 'CBIQ',
 ↪  'LDB0431', 'LDB0432', 'LDB0435', 'LDB0438', 'LDB0447', 'LDB0448', 'LDB0453', 'NADE',
 ↪  'LDB0482', 'PEPD1', 'UDK1', 'GLNH1', 'GLNP', 'PPX', 'LDB0533', 'LDB0541', 'FLAV',
 ↪  'LDB0563', 'LDB0567', 'PTSH', 'PTSI', 'SPX', 'HPF', 'UVRB', 'UVRA', 'LDB0631', 'CGGR',
 ↪  'GAP', 'POTD1', 'LDB0671', 'LDB0675', 'COMGC', 'ACK', 'LDB0689', 'LDB0691', 'HEMK',
 ↪  'MREB2', 'FTSA', 'FTSZ', 'LDB0754', 'RECD', 'RNJ', 'LDB0808', 'LDB0812', 'LDB0827',
 ↪  'LDB0844', 'LDB0849', 'LDB0861', 'LDB0877', 'FABF', 'ACCC', 'LDB0930', 'PHOU', 'LDB0967',
 ↪  'LDB0977', 'LDB0980', 'MVAD', 'DNAD', 'NTH', 'LDB1015', 'LDB1019', 'LSP', 'LDB1025',
 ↪  'LDB1031', 'LDB1099', 'LDB1087']
['TETM', 'RPOA', 'ATPD', 'RECA', 'HSP60', 'TUF', 'DNAK', 'PHES', 'HISS', 'LEUS', 'RPOB',
 ↪  'GND', 'RPLB', 'CK', 'ADI', 'HDCA', 'GLDA', 'GLDH', 'SLPLH1', 'HDCC', 'HDCB', 'GROEL',
 ↪  'FAEB', 'FAEA', 'OTC', 'GLDE', 'HDCP', 'ABC5', 'MLE', 'RECN', 'GLDD', 'GLDG', 'GLDK',
 ↪  'GLDB', 'GLDC', 'HISRS', 'GTF', 'YVYE', 'SECA', 'RF2', 'ORFX', 'HDHA', 'PEPQ', 'CLPE',
 ↪  'CLPP', 'DSY26_01720', 'DSY26_01985', 'DSY26_01415', 'GALU', 'DSY26_00135',
 ↪  'DSY26_00025', 'DSY26_02640', 'DSY26_02455', 'DSY26_03410', 'DSY26_03165', 'FABZ',
 ↪  'DSY26_01640', 'DSY26_01520', 'DSY26_05985', 'DSY26_01330', 'DSY26_06325', 'DSY26_02175',
 ↪  'DSY26_02125', 'DSY26_06155', 'DSY26_01945', 'DSY26_06660', 'DSY26_01840', 'DSY26_01535',
 ↪  'DSY26_01465', 'CDD', 'TYPA', 'DSY26_07045', 'DSY26_01225', 'DSY26_07555', 'DSY26_01105',
 ↪  'DSY26_07710', 'PGSA', 'DSY26_00880', 'DSY26_02255', 'DSY26_08090', 'DSY26_08220',
 ↪  'DSY26_02485', 'DSY26_02415', 'DSY26_02355', 'DSY26_01955', 'DSY26_08415', 'NRDJ',
 ↪  'DSY26_00270', 'DSY26_02180', 'DSY26_01610', 'DSY26_03155', 'DSY26_03435', 'ASNC',
 ↪  'DSY26_01745', 'DSY26_01875', 'DSY26_00950', 'DSY26_02730', 'DSY26_02675', 'DSY26_00770',
 ↪  'DSY26_02565']
```

1.2.3 Substitution rates

Script 1.2.5 (python)

```python
#Proteomes in fasta to dictionaries
RESTART = True
VERBOSE = True
#Protein limit
LIMIT = 20

if RESTART:
    proteome_termophilus = load("proteome_termophilus")
    proteome_streptococcus = load("proteome_streptococcus")

# Compute keys to process: only the keys that are included in both groups
proteome_keys = []
for key in proteome_termophilus.keys():
    if key in proteome_streptococcus.keys():
```

```
        proteome_keys.append(key)

if VERBOSE: print(proteome_keys[0:LIMIT])
print("Proteins we need to process:", len(proteome_keys))
limit =  min(len(proteome_keys), LIMIT)
print("Proteins we want to process:", limit)

# Compute branch lengths
subst_rates_groups = {}
subst_rates_groups["termophilus"] = compute_subst_rates(proteome_termophilus,
    proteome_keys[0:limit],
                                                        "termophilus", False)
subst_rates_groups["streptococcus"] = compute_subst_rates(proteome_streptococcus,
    proteome_keys[0:limit],
                                                          "streptococcus", False)

dump(subst_rates_groups, "subst_rates_groups")
if VERBOSE: print(subst_rates_groups)
```

Output

```
['TMRNA', 'GALT', 'LACZ', 'SBCD', 'GALR', 'DNAN', 'FTSH', 'MREC', 'ARAT', 'PURL', 'TAG',
    'LABC', 'ILVA', 'CBIM', 'TRKH2', 'DNAB', 'SERS', 'LIVM', 'HPF', 'FABF']
Proteins we need to process: 964
Proteins we want to process: 20
{'termophilus': {'TMRNA': -1, 'GALT': 0.014773333333333333, 'LACZ': 0.00292, 'SBCD':
    0.00245625, 'FTSH': 0.00262, 'ILVA': 0.0024, 'SERS': 0.001318888888888889, 'HPF':
    0.01282, 'FABF': 0.31672}, 'streptococcus': {'TMRNA': 0.10939888888888888, 'GALT':
    0.011721821086261982, 'LACZ': 0.028357828947368432, 'SBCD': 0.05702195652173911, 'GALR':
    0.0795148, 'DNAN': 0.01873888888888889, 'FTSH': 0.017297005347593573, 'MREC':
    0.06296962962962963, 'ARAT': 0.17196846153846151, 'PURL': 0.012450666666666667, 'TAG':
    0.02267730769230769, 'ILVA': 0.012551562499999992, 'CBIM': 0.10676333333333335, 'TRKH2':
    0.16780555555555557, 'DNAB': 0.09844512820512821, 'SERS': 0.008190401785714287, 'LIVM':
    0.027198333333333328, 'HPF': 0.03110622641509432, 'FABF': 0.029105231788079465}}
```

Script 1.2.6 (python)

```
#Proteomes in fasta to dictionaries
RESTART = True
VERBOSE = True
#Protein limit
LIMIT = 50

if RESTART:
    proteome_bulgaricus = load("proteome_bulgaricus")
    proteome_lactobacillus = load("proteome_lactobacillus")

# Compute keys to process: only the keys that are included in both groups
proteome_keys = []
for key in proteome_bulgaricus.keys():
    if key in proteome_lactobacillus.keys():
```

```
        proteome_keys.append(key)

if VERBOSE: print(proteome_keys[0:LIMIT])
print("Proteins we need to process:", len(proteome_keys))
limit =  min(len(proteome_keys), LIMIT)
print("Proteins we want to process:", limit)

# Compute branch lengths
subst_rates_groups_lacto = {}
subst_rates_groups_lacto["bulgaricus"] = compute_subst_rates(proteome_bulgaricus,
    proteome_keys[0:limit],
                                                             "bulgaricus", False)
subst_rates_groups_lacto["lactobacillus"] = compute_subst_rates(proteome_lactobacillus,
    proteome_keys[0:limit],
                                                               "lactobacillus", False)

dump(subst_rates_groups_lacto, "subst_rates_groups_lacto")
if VERBOSE: print(subst_rates_groups_lacto)
```

Output

```
['TMRNA', 'EXOA', 'PHNB', 'NRDG', 'CSHA', 'FTSH', 'LYSS', 'CBIQ', 'NADE', 'PEPD1', 'GLNH1',
    'GLNP', 'PPX', 'PTSH', 'PTSI', 'SPX', 'HPF', 'UVRB', 'UVRA', 'CGGR', 'GAP', 'COMGC',
    'HEMK', 'FTSA', 'FTSZ', 'RECD', 'RNJ', 'FABF', 'ACCC', 'PHOU', 'MVAD', 'DNAD', 'NTH',
    'LSP', 'MOD', 'POLC', 'OPPB', 'RECG', 'RPMA', 'PURF', 'PURS', 'CFA', 'PHET', 'PRSA',
    'REX', 'PYRR2', 'PEPC', 'THIJ', 'THRC', 'GLPQ']
Proteins we need to process: 664
Proteins we want to process: 50
{'bulgaricus': {'TMRNA': -1, 'CSHA': 0.01663, 'FTSH': 0.0023866666666666667, 'LYSS': 0.01397,
    'NADE': 0.009055, 'HPF': 0.00541, 'UVRB': 0.0040425, 'UVRA': 0.0035033333333333336,
    'FTSA': 0.011955, 'FTSZ': 0.0060825, 'RNJ': 0.06184272727272726, 'NTH': 0.00478, 'POLC':
    0.0029900000000000005, 'RECG': 0.0024533333333333334, 'RPMA': 0.01003, 'PURF':
    0.0038640000000000002, 'PURS': 0.02439, 'PHET': 0.00934, 'PRSA': 0.010936666666666666,
    'REX': 0.022005}, 'lactobacillus': {'TMRNA': 0.09091, 'EXOA': 0.1424788888888889, 'NRDG':
    0.04172188679245283, 'CSHA': 0.029512741935483876, 'FTSH': 0.017883125000000003, 'LYSS':
    0.015703070175438597, 'CBIQ': 0.06663888888888889, 'NADE': 0.020291111111111117, 'PPX':
    0.08499444444444443, 'PTSH': 0.10795666666666666, 'PTSI': 0.09214411764705883, 'HPF':
    0.040429199999999985, 'UVRB': 0.0182108527131783, 'UVRA': 0.01776475, 'CGGR':
    0.11341333333333334, 'GAP': 0.04136581395348837, 'HEMK': 0.196304, 'FTSA':
    0.030103584905660376, 'FTSZ': 0.029649259259259254, 'RECD': 0.11507000000000002, 'RNJ':
    0.018687327188940086, 'FABF': 0.030076666666666675, 'ACCC': 0.039767567567567566, 'PHOU':
    0.07015690909090912, 'MVAD': 0.05137633802816899, 'DNAD': 0.2577, 'NTH':
    0.03143635416666666, 'POLC': 0.01936439189189189, 'OPPB': 0.14599, 'RECG':
    0.01724738853503185, 'RPMA': 0.06790682926829267, 'PURF': 0.015611698113207553, 'PURS':
    0.03728014084507042, 'CFA': 0.15947, 'PHET': 0.0215886, 'PRSA': 0.03705391025641024,
    'REX': 0.028940109890109877, 'PEPC': 0.04863730769230771, 'THRC': 0.228004, 'GLPQ':
    0.38304333333333335}}
```

1.3 Selection of first set of target proteins

At this step is necessary to calculate the ratios, the means of the ratios and select the more extreme between them (two tailed percentile 80%).

1.3.1 Methods

Script 1.3.1 (python)

```python
# Compute branch ratios
# Compute mean of branch ratios and standard deviation
# Obtain the most extreme values.
# These are the proteins that could have been a slowdown or from his initial state
import statistics as stats

def compute_branch_ratios(subst_rates_groups, group1, group2, verbose=False):
    """
    For every protein in group1 that has counterpart in group2 calculate the ratio of
    branch lengths.

    Returns:
        dict of string, float: ratios by protein
    """
    ratios = {}
    for protein in subst_rates_groups[group1].keys():
        if protein in subst_rates_groups[group2].keys():
            ratio = subst_rates_groups[group1][protein]/(subst_rates_groups[group2][protein]
            ↪  + 0.00001)
            if subst_rates_groups[group1][protein] != -1 and
            ↪  subst_rates_groups[group2][protein] != -1:
                ratios[protein] = ratio
            if verbose: print(group1, protein,
                              subst_rates_groups[group1][protein],
                              subst_rates_groups[group2][protein],
                              ratio)
    return ratios

def compute_mean_std(ratios, verbose=False):
    """
    Calculate mean and std for ratios
    """
    ratios_list =  list(ratios.values())
    if verbose: print("ratios_list",ratios_list)
    ratios_mean = stats.mean(ratios_list)
    ratios_stdev = stats.stdev(ratios_list)
    return ratios_mean, ratios_stdev

def filter_target_proteins(ratios, mean, std, n_std, verbose=False):
    """
    Filter proteins that are n_std > mean or n_std < mean
    """
    proteins = []
    for protein in ratios.keys():
```

```
        if verbose: print(protein, ratios[protein], mean, std)
        if ratios[protein] > mean + n_std * std or ratios[protein] < mean - n_std * std:
            proteins.append(protein)
    return proteins
```

1.3.2 Selection of target proteins

Script 1.3.2 (python)

```
RESTART = True
VERBOSE = False

if RESTART:
    subst_rates_groups = load("subst_rates_groups")

rat = compute_branch_ratios(subst_rates_groups, "termophilus", "streptococcus", VERBOSE)
if VERBOSE: print(rat)

if len(rat) > 1:
    mean, std = compute_mean_std(rat, VERBOSE)
    if VERBOSE: print("\nMean", mean, "Standard deviation", std, "\n")
    prot = filter_target_proteins(rat, mean, std, 0.8, VERBOSE)
    print("Target proteins Streptococcus:", prot)
else:
    print("No results: protein input number < 2")
```

Output

```
Target proteins: ['FABF']
```

Script 1.3.3 (python)

```
RESTART = True
VERBOSE = True

if RESTART:
    subst_rates_groups_lacto = load("subst_rates_groups_lacto")

rat = compute_branch_ratios(subst_rates_groups_lacto, "bulgaricus", "lactobacillus", VERBOSE)
if VERBOSE: print(rat)

if len(rat) > 1:
    mean, std = compute_mean_std(rat, VERBOSE)
    if VERBOSE: print("\nMean", mean, "Standard deviation", std, "\n")
    prot = filter_target_proteins(rat, mean, std, 0.8, VERBOSE)
    print("Target proteins Lactobacillus:", prot)
else:
    print("No results: protein input number < 2")
```

Output

```
bulgaricus TMRNA -1 0.09091 -10.998680158380994
bulgaricus CSHA 0.01663 0.029512741935483876 0.5632945624204412
bulgaricus FTSH 0.0023866666666666667 0.017883125000000003 0.13338456343800573
bulgaricus LYSS 0.01397 0.015703070175438597 0.8890687716673459
bulgaricus NADE 0.009055 0.020291111111111117 0.4460346997974932
bulgaricus HPF 0.00541 0.040429199999999985 0.13378108370096345
bulgaricus UVRB 0.0040425 0.0182108527131783 0.22186118639092267
bulgaricus UVRA 0.0035033333333333336 0.01776475 0.19709606792406834
bulgaricus FTSA 0.011955 0.030103584905660376 0.3969969048006918
bulgaricus FTSZ 0.0060825 0.029649259259259254 0.20507929570429576
bulgaricus RNJ 0.06184272727272726 0.018687327188940086 3.307570469714447
bulgaricus NTH 0.00478 0.03143635416666666 0.1520049025291088
bulgaricus POLC 0.0029900000000000005 0.01936439189189189 0.15432742440041716
bulgaricus RECG 0.0024533333333333334 0.01724738853503185 0.14216133155680877
bulgaricus RPMA 0.01003 0.06790682926829267 0.1476806280278246
bulgaricus PURF 0.0038640000000000002 0.015611698113207553 0.24734826982305688
bulgaricus PURS 0.02439 0.03728014084507042 0.6540602810092159
bulgaricus PHET 0.00934 0.0215886 0.43243543563008713
bulgaricus PRSA 0.010936666666666666 0.03705391025641024 0.29507589973659504
bulgaricus REX 0.022005 0.028940109890109877 0.7601007417079785
{'CSHA': 0.5632945624204412, 'FTSH': 0.13338456343800573, 'LYSS': 0.8890687716673459, 'NADE':
 ↪  0.4460346997974932, 'HPF': 0.13378108370096345, 'UVRB': 0.22186118639092267, 'UVRA':
 ↪  0.19709606792406834, 'FTSA': 0.3969969048006918, 'FTSZ': 0.20507929570429576, 'RNJ':
 ↪  3.307570469714447, 'NTH': 0.1520049025291088, 'POLC': 0.15432742440041716, 'RECG':
 ↪  0.14216133155680877, 'RPMA': 0.1476806280278246, 'PURF': 0.24734826982305688, 'PURS':
 ↪  0.6540602810092159, 'PHET': 0.43243543563008713, 'PRSA': 0.29507589973659504, 'REX':
 ↪  0.7601007417079785}
ratios_list [0.5632945624204412, 0.13338456343800573, 0.8890687716673459, 0.4460346997974932,
 ↪  0.13378108370096345, 0.22186118639092267, 0.19709606792406834, 0.3969969048006918,
 ↪  0.20507929570429576, 3.307570469714447, 0.1520049025291088, 0.15432742440041716,
 ↪  0.14216133155680877, 0.1476806280278246, 0.24734826982305688, 0.6540602810092159,
 ↪  0.43243543563008713, 0.29507589973659504, 0.7601007417079785]

Mean 0.4989138168410404 Standard deviation 0.7176540409439205

CSHA 0.5632945624204412 0.4989138168410404 0.7176540409439205
FTSH 0.13338456343800573 0.4989138168410404 0.7176540409439205
LYSS 0.8890687716673459 0.4989138168410404 0.7176540409439205
NADE 0.4460346997974932 0.4989138168410404 0.7176540409439205
HPF 0.13378108370096345 0.4989138168410404 0.7176540409439205
UVRB 0.22186118639092267 0.4989138168410404 0.7176540409439205
UVRA 0.19709606792406834 0.4989138168410404 0.7176540409439205
FTSA 0.3969969048006918 0.4989138168410404 0.7176540409439205
FTSZ 0.20507929570429576 0.4989138168410404 0.7176540409439205
RNJ 3.307570469714447 0.4989138168410404 0.7176540409439205
NTH 0.1520049025291088 0.4989138168410404 0.7176540409439205
POLC 0.15432742440041716 0.4989138168410404 0.7176540409439205
RECG 0.14216133155680877 0.4989138168410404 0.7176540409439205
RPMA 0.1476806280278246 0.4989138168410404 0.7176540409439205
PURF 0.24734826982305688 0.4989138168410404 0.7176540409439205
PURS 0.6540602810092159 0.4989138168410404 0.7176540409439205
```

```
PHET 0.43243543563008713 0.4989138168410404 0.7176540409439205
PRSA 0.29507589973659504 0.4989138168410404 0.7176540409439205
REX 0.7601007417079785 0.4989138168410404 0.7176540409439205
Target proteins Lactobacillus: ['RNJ']
```

1.4 Final thoughts

At this stage we make a explanation of the results, justifications and propose a better computational approach.
It's not mandatory that we have confidence on this explanation, but it should seem credible.

2 Generate document outputs.

Script 2.0.1 (text)

```
%%bash
#cd /Users/nandoide/Desktop/uni/STRBI.practical
jupyter nbconvert --to=latex --template=~/report.tplx coevolution.ipynb 1> /dev/null
pdflatex -shell-escape coevolution 1> /dev/null
```

Output

```
[NbConvertApp] Converting notebook coevolution.ipynb to latex
[NbConvertApp] Writing 67023 bytes to coevolution.tex
```