The Double Life of RNA: Uncovering Non-Coding RNAs

Erik S. Wright

April 26, 2022

Contents

1 Introduction

RNA leads a double life as both messenger (mRNA) and an alphabet soup of roles in the cell (e.g., rRNA, tRNA, snRNA, tmRNA, etc.). RNAs that take on a life of their own are notoriously difficult to detect *ab initio* using bioinformatics. This is in part because their signature characteristic, that a single strand of RNA folds into a stable structure, can be found in many regions of the genome. This makes life difficult because there is no clear signature that a genomic region contains a non-coding RNA in the way there is for protein coding regions. The most promising approach is to search for intergenic regions with high GC-content a folded structure that is conserved across many genomes. But what do you do if you only have a single genome and want to find non-coding RNAs? That is exactly what this vignette shows you how to do!

Before we can have the time of our lives finding non-coding RNAs, we must first train models on examples of known RNAs. As a case study, this tutorial focuses on finding all known non-coding RNAs in the genome of *Chlamydia trachomatis*, an intracellular bacterial pathogen known for causing chlamydia. This genome was chosen because it is relatively small (only 1 Mbp) so the examples run quickly. *Chlamydia* harbors all of the standard non-coding RNAs that date back near the origin of life (i.e., rRNAs, tRNAs, etc.). It also has one small RNA (sRNA), named IhtA, that has only been found in Chlamydiae and is believed to play a role in regulating its life stages. We are going to find IhtA in the genome of *C. trachomatis*, then we are going to use the same approach to find all of the standard non-coding RNAs that are conserved across many forms of life.

2 Getting Started

2.1 Startup

To get started we need to load the DECIPHER package, which automatically loads a few other required packages.

```
> library(DECIPHER)
```

Searches for non-coding RNAs are performed by the function `FindNonCoding`. Help can be accessed via:

```
> ? FindNonCoding
```

Once DECIPHER is installed, the code in this tutorial can be obtained via:

```
> browseVignettes("DECIPHER")
```

3 Building a Non-Coding RNA Model

Before we can find a non-coding RNA, we need to create a multiple sequence alignment of some of its sequence representatives. This alignment will be the input to `LearnNonCoding`.

3.1 Importing the sequences

The first step is to set filepaths to the sequences (in FASTA format). In this case we are going to use the IhtA sequences included with DECIPHER, but you could follow along with your own set of homologous sequences. Be sure to change the path names to those on your system by replacing all of the text inside quotes labeled "<<path to ...>>" with the actual path on your system.

```
> # specify the path to your genome:
> fas_path <- "<<path to FASTA file>>"
> # OR use the example genome:
> fas_path <- system.file("extdata",
          "IhtA.fas",
          package="DECIPHER")
> # read the sequences into memory
> rna <- readRNAStringSet(fas_path)
> rna
RNAStringSet object of length 27:
     width seq                                                names
 [1]   103 AAGAUGAUAUUCUGCGCCAUGGA...UGAUUGUUGUUUUCUUGGCUUU KE360988.1/6443-6...
 [2]   105 AAGAUGAUAUUCUCCGCCGUGGA...AACUUUAUGUUUUCUUGGCUUU AE002161.1/54676-...
 [3]   107 AAGUUGGUAUUCUAACGCCAUGG...UAUCUCCGGUUCUCUUGGCUUU AE001273.1/773281...
 [4]   107 AAGUUGGUAUUCUAACGCCAUGG...UGUCUCCAGUUCUCUUGGCUUU AE002160.2/53054-...
 [5]   103 AAGAUGAUAUUCUACGCCAUGGA...UGAUUGUUGUUUUCUUGGCUUU AE015925.1/52097-...
 ...   ... ...
[23]   107 AAGUUGGUAUUCUAACGCCAUGG...UAUCUCCGGUUCUCUUGGCUUU CP006674.1 Chlamy...
[24]   103 AAGAUGAUAUUCUGCGCCAUGGA...UGAUUGUUGUUUUCUUGGCUUU KE356008.1 Chlamy...
[25]   108 AAGUUGGUAUUCUAACGCCAUGG...UAUCUCCGGUUCUCUUGGCUUU CVNC01000001.1 Ch...
[26]   103 AAGAUGAUAUUCUGCGCCAUGGA...UGAUUGUUGUUUUCUUGGCUUU AP006861.1 Chlamy...
[27]   101 AAGAUGAUAUUCUACGCCAUGGA...AUGAUGUUGUUUUCUUGGCUUU CP015840.1 Chlamy...
```

Ideally we would start with a few thousand diverse sequence representatives, yet only 27 representatives of IhtA are known. That's life, so we will have to make due with what we have available.

3.2 Aligning the sequences

Next we need to align the sequences with `AlignSeqs`. Note that non-coding RNA alignments are more accurate if we provide the sequences as a *RNAStringSet* rather than the equivalent *DNAStringSet*, because `AlignSeqs` will use conserved secondary structure to improve the alignment. Alignment is fast, so hold on for dear life! (Consider adding `processors=NULL` if you want it to go even faster with multiple processors.)

```
> RNA <- AlignSeqs(rna)
Determining distance matrix based on shared 7-mers:
================================================================================

Time difference of 0.01 secs

Clustering into groups by similarity:
================================================================================

Time difference of 0.02 secs

Aligning Sequences:
================================================================================

Time difference of 0.31 secs

Iteration 1 of 2:

Determining distance matrix based on alignment:
================================================================================

Time difference of 0 secs

Reclustering into groups by similarity:
================================================================================

Time difference of 0.02 secs

Realigning Sequences:
================================================================================

Time difference of 0.17 secs

Iteration 2 of 2:

Determining distance matrix based on alignment:
================================================================================

Time difference of 0 secs

Reclustering into groups by similarity:
================================================================================

Time difference of 0.02 secs
```

```
Realigning Sequences:
==========================================================================

Time difference of 0.03 secs
> RNA
RNAStringSet object of length 27:
     width seq                                                 names
 [1]   113 AAGAUGAUAUUCU-GCGCCAUGG...GAUUGU-UGUUUUCUUGGCUUU KE360988.1/6443-6...
 [2]   113 AAGAUGAUAUUCU-CCGCCGUGG...AACUUUAUGUUUUCUUGGCUUU AE002161.1/54676-...
 [3]   113 AAGUUGGUAUUCUAACGCCAUGG...--UCUCCGGUUCUCUUGGCUUU AE001273.1/773281...
 [4]   113 AAGUUGGUAUUCUAACGCCAUGG...--UCUCCAGUUCUCUUGGCUUU AE002160.2/53054-...
 [5]   113 AAGAUGAUAUUCU-ACGCCAUGG...GAUUGU-UGUUUUCUUGGCUUU AE015925.1/52097-...
 ...   ... ...
[23]   113 AAGUUGGUAUUCUAACGCCAUGG...--UCUCCGGUUCUCUUGGCUUU CP006674.1 Chlamy...
[24]   113 AAGAUGAUAUUCU-GCGCCAUGG...GAUUGU-UGUUUUCUUGGCUUU KE356008.1 Chlamy...
[25]   113 AAGUUGGUAUUCUAACGCCAUGG...--UCUCCGGUUCUCUUGGCUUU CVNC01000001.1 Ch...
[26]   113 AAGAUGAUAUUCU-GCGCCAUGG...GAUUGU-UGUUUUCUUGGCUUU AP006861.1 Chlamy...
[27]   113 AAGAUGAUAUUCU-ACGCCAUGG...GA-UGU-UGUUUUCUUGGCUUU CP015840.1 Chlamy...
```

We can see from the alignment that the sequences have both conserved and variable regions. But if we really want to bring the sequences to life we need to look at their predicted secondary structures. The IhtA is believed to form three back-to-back hairpin loops based on its minimum free energy structure. We can use `PredictDBN` to predict the secondary structures of the sequences.

```
> p <- PredictDBN(RNA, type="structures")
Determining distance matrix based on alignment:
==========================================================================

Time difference of 0 secs

Determining sequence weights:
==========================================================================

Time difference of 0.02 secs

Computing Free Energies:
==========================================================================

Time difference of 0.63 secs

Predicting RNA Secondary Structures:
==========================================================================

Time difference of 0.01 secs
> BrowseSeqs(RNA, patterns=p)
```

In this color scheme, blue regions pair to green regions and red regions are unpaired. There is clear evidence for a hairpin loop near the 3'-end if IhtA, and weaker evidence for conserved secondary structure elsewhere in the sequences (Fig. 1).

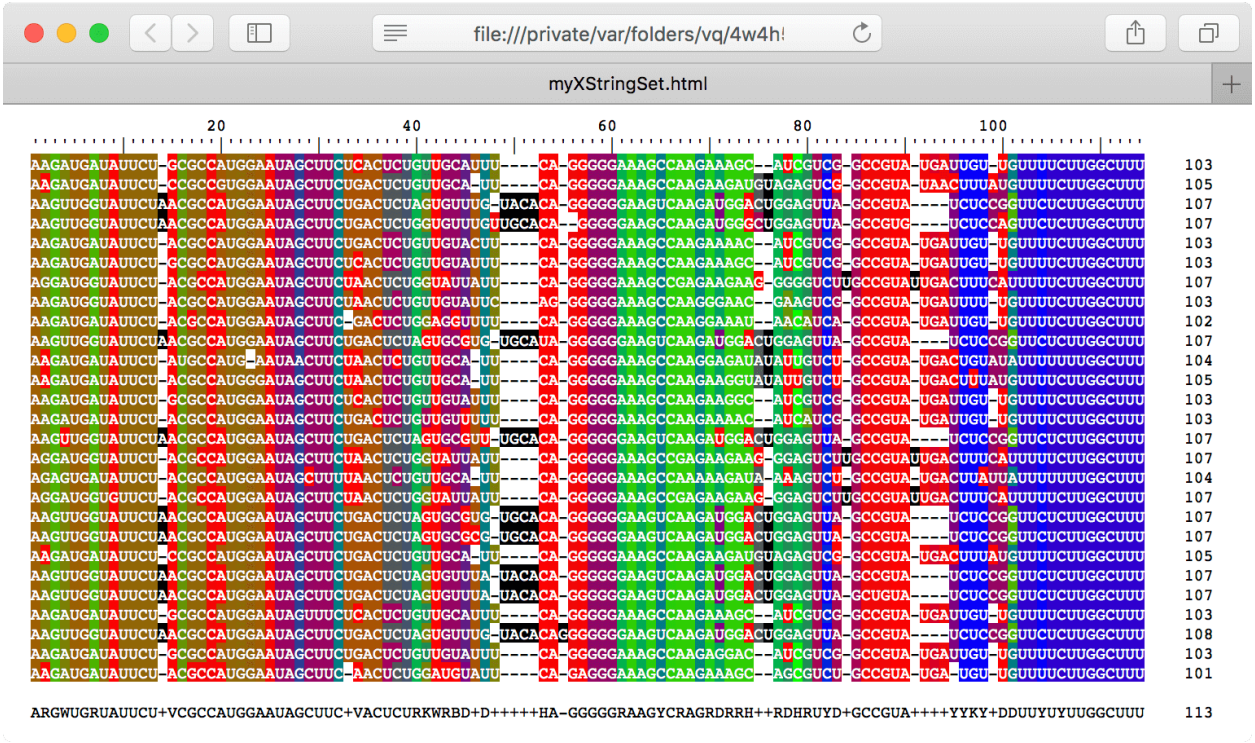

Figure 1: Predicted secondary structure of IhtA

We can also visualize the secondary structure through a dot plot (Fig. 2). One half of the dot plot shows the probabilities of pairing, and the other half shows the predicted structure. This view reveals the level of ambiguity in the secondary structure based on the (low) amount of available information.

```
> evidence <- PredictDBN(RNA, type="evidence", threshold=0, verbose=FALSE)
> pairs <- PredictDBN(RNA, type="pairs", verbose=FALSE)
> dots <- matrix(0, width(RNA)[1], width(RNA)[1])
> dots[evidence[, 1:2]] <- evidence[, 3]
> dots[pairs[, 2:1]] <- 1
> image(dots, xaxt="n", yaxt="n", col=gray(seq(1, 0, -0.01)))
> abline(a=0, b=1)
> cons <- toString(ConsensusSequence(RNA, threshold=0.2))
> cons <- strsplit(cons, "")[[1]]
> at <- seq(0, 1, length.out=length(cons))
> axis(1, at, cons, tick=FALSE, cex.axis=0.3, gap.axis=0, line=-1)
> axis(2, at, cons, tick=FALSE, cex.axis=0.3, gap.axis=0, line=-1)
```

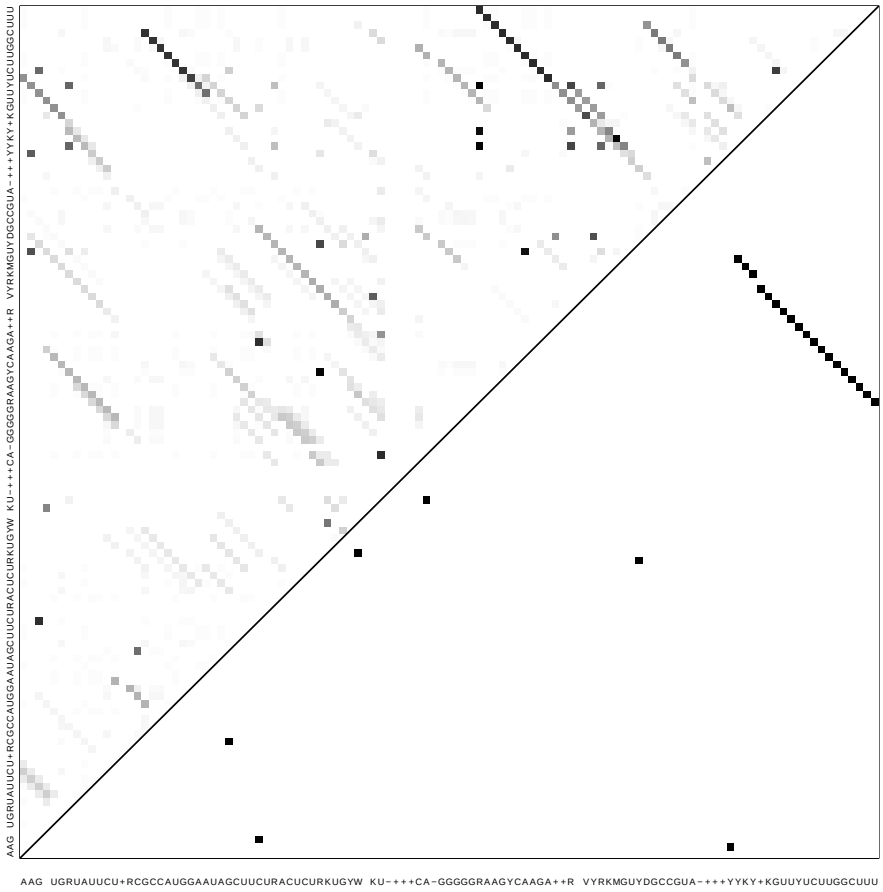

Figure 2: Secondary Structure Dot Plot.

Two considerations are important at this stage: (*i*) that we have a clearly defined boundary representing the true beginning and end of every sequence, and (*ii*) that the sequences are a diverse sample of what we hope to find. Here we don't have any partial sequences, but if we did they could be identified by counting gaps ("-") at their ends with `TerminalChar`. Any partial sequences should be removed before proceeding, for example by using:

```
> RNA <- unique(RNA)
> t <- TerminalChar(RNA)
> w <- which(t[, "leadingChar"] <= median(t[, "leadingChar"]) &
         t[, "trailingChar"] <= median(t[, "trailingChar"]))
> RNA <- RemoveGaps(RNA[w], "common")
```

Rather than remove partial sequences, it would have been possible to shorten the alignment to the region shared by all sequences using `subseq`.

3.3 Learning sequence patterns

Now we need to build a model capturing the essential characteristics of the non-coding RNA. The function `LearnNonCoding` takes an alignment as input and outputs an object of class *NonCoding* that describes the sequences.

```
> y <- LearnNonCoding(RNA)
> y
NonCoding object with 16 motifs, 3 hairpins, and 2-mer frequencies.
```

The output object is a list containing patterns of three types: "motifs", "hairpins", and "kmers". Motifs are short regions of the sequence that can be used to identify the sequences:

```
> y[["motifs"]]
   begin_low begin_high end_low end_high         motif          pwm
1          0          0      90       94 AAGaTGrTATTCT 0.945645....
2         13         14      82       86      ACGCCATG 0.796028....
3         21         22      81       85             G 0.018118....
4         22         23      72       76     AATAGCTTC 0.911404....
5         31         33      59       63  rACTCTGkTGyw 0.463332....
6         44         46      56       60            TT 0.042292....
7         46         51      54       58            CA 0.083772....
8         48         53      48       52        GGGGGA 0.018118....
9         54         59      39       43     AAGCCAAGA 0.945645....
10        64         70      34       38           rwr 0.589327....
11        68         74      20       24 rdmGTydGCCGTA 0.554448....
12        82         86      19       20             A 0.945645....
13        84         87      18       19             y 0.018452....
14        85         88      15       16           Tkt 0.018452....
15        88         92       8        8       tGTTTC 0.164952....
16        95         99       0        0      TTGGCTTT 0.018118....
       minscore   prevalence   background
1  0, 10.50.... 0.035714.... 0.996258....
2  0, 4.732.... 0.035714.... 0.988400....
3        0, Inf 0.053671.... 0.551778....
4  0, 7.963.... 0.053671.... 0.981026....
5  0, 6.745.... 0.035714.... 0.975771....
6  0.731723.... 0.120251.... 0.706004....
7        0, Inf 0.064669.... 0.462564....
```

```
8  0, 6.420.... 0.064669.... 0.903487....
9  0, 9.116.... 0.064669.... 0.978091....
10 1.471911.... 0.396535.... 0.688852....
11 0, 7.172.... 0.035714.... 0.972067....
12       0, Inf 0.110082.... 0.581909....
13 0.486998.... 0.064669.... 0.447887....
14 0, 0.560.... 0.035714.... 0.638935....
15 0, 4.914.... 0.035714.... 0.947343....
16       0, Inf 0.035714.... 0.990614....
```

Note that some of the motifs contain ambiguity codes (see `IUPAC_CODE_MAP`) that represent multiple nucleotides. Motifs are defined by their distance from each end of the non-coding RNA, and their prevalence across sequence representatives when allowing for a certain degree of distance.

```
> y[["hairpins"]]
  begin_low begin_high end_low end_high width_low width_high length_low
1        -4          1      63       72        31         44          6
2        30         37      24       27        43         51          6
3        49         58      -4        0        49         61         13
  length_high           dG   prevalence   background
1          12 -Inf, -2.... 0.308347.... 0.208862....
2          15 -Inf, -1.... 0.067775.... 0.000315....
3          26 -Inf, -2.... 0.137634.... 0.000105....
```

Hairpins are defined similarly, but allow for ambiguity in the form of varying free energy (dG). As we saw in the predicted structures, the IhtA sequences end in a prominent hairpin that is both long and has a low free energy.

```
> head(y[["kmers"]])
[1] 186 109 182 163  78  83
> tail(y[["kmers"]])
[1] 236 220 216 218 154 341
```

Finally, the sequences are identifiable by their k-mer frequencies. In general, non-coding RNAs have higher GC-content than protein coding regions of genomes. Note that the value of "k" is set automatically depending on the amount of information in the input sequence alignment.

4 Finding Non-Coding RNAs

Now that we have captured the life force of the sequences, our next goal is to find homologous non-coding RNAs in a genome. You can either use your own genome or follow along with the example *C. trachomatis* genome.

```
> # specify the path to your genome:
> genome_path <- "<<path to genome FASTA file>>"
> # OR use the example genome:
> genome_path <- system.file("extdata",
          "Chlamydia_trachomatis_NC_000117.fas.gz",
          package="DECIPHER")
> # read the sequences into memory
> genome <- readDNAStringSet(genome_path)
> genome
```

```
DNAStringSet object of length 1:
      width seq                                                 names
[1] 1042519 GCGGCCGCCCGGGAAATTGCTA...GTTGGCTGGCCCTGACGGGGTA NC_000117.1 Chlam...
```

The function `FindNonCoding` finds matches to *NonCoding* models in a genome. Let's search for the IhtA model in the *Chlamydia* genome:

```
> FindNonCoding(y, genome)
================================================================================

Time difference of 2.1 secs
Genes object of size 1 specifying:
1 non-coding RNA of 107 nucleotides.

  Index Strand  Begin    End TotalScore Gene
1     1      0 773281 773387      93.29   -1
```

And there it is! The output tells us that the IhtA gene is found on the forward strand of the first sequence (`"Index"`) in the genome. This match had a high score to first (and only) model in `y` (i.e., `"Gene"` is `-1`). Values in the `"Gene"` column are negative to signify that these are non-coding RNAs and not protein coding genes.

Life's too short to build models for every non-coding RNA, so we can load a set of pre-built models for our bacterial genome. Replace "Bacteria" with "Archaea" or "Eukarya" for genomes from organisms belonging to other domains of life.

```
> data(NonCodingRNA_Bacteria)
> x <- NonCodingRNA_Bacteria
> names(x)
 [1] "tRNA-Ala"
 [2] "tRNA-Arg"
 [3] "tRNA-Asn"
 [4] "tRNA-Asp"
 [5] "tRNA-Cys"
 [6] "tRNA-Gln"
 [7] "tRNA-Glu"
 [8] "tRNA-Gly"
 [9] "tRNA-His"
[10] "tRNA-Ile"
[11] "tRNA-Leu"
[12] "tRNA-Lys"
[13] "tRNA-Met"
[14] "tRNA-Phe"
[15] "tRNA-Pro"
[16] "tRNA-Ser"
[17] "tRNA-Thr"
[18] "tRNA-Trp"
[19] "tRNA-Tyr"
[20] "tRNA-Val"
[21] "tRNA-Sec"
[22] "rRNA_5S-RF00001"
[23] "rRNA_16S-RF00177"
[24] "rRNA_23S-RF02541"
[25] "tmRNA-RF00023"
```

```
[26] "tmRNA_Alpha-RF01849"
[27] "RNase_P_class_A-RF00010"
[28] "RNase_P_class_B-RF00011"
[29] "SsrS-RF00013"
[30] "Intron_Gp_I-RF00028"
[31] "Intron_Gp_II-RF00029"
[32] "SmallSRP-RF00169"
[33] "Cyclic-di-GMP_Riboswitch-RF01051"
[34] "Cyclic-di-AMP_Riboswitch-RF00379"
[35] "T-box_Leader-RF00230"
[36] "Ribosomal_Protein_L10_Leader-RF00557"
[37] "Cobalamin_Riboswitch-RF00174"
[38] "TPP_Riboswitch-RF00059"
[39] "SAM_Riboswitch-RF00162"
[40] "Fluoride_Riboswitch-RF01734"
[41] "FMN_Riboswitch-RF00050"
[42] "Glycine_Riboswitch-RF00504"
[43] "HEARO-RF02033"
[44] "Flavo_1-RF01705"
[45] "Acido_Lenti_1-RF01687"
[46] "5'_ureB-RF02514"
```

What a life saver! Our new dataset (`x`) is a list of models, including tRNAs (by amino acid), transfer-messenger RNA, RNase P, SsrS (6S RNA), group I and II introns, the signal recognition particle, and three rRNAs (5S, 16S, and 23S). Let's add the model we built of IhtA into the list:

```
> x[[length(x) + 1]] <- y
> names(x)[length(x)] <- "IhtA"
```

Now we can search for them all at once with `FindNonCoding`:

```
> rnas <- FindNonCoding(x, genome)
================================================================================

Time difference of 73.02 secs
> rnas
Genes object of size 47 specifying:
47 non-coding RNAs from 72 to 2,938 nucleotides.

  Index Strand  Begin    End TotalScore Gene
1     1      1  20663  21082      88.33  -25
2     1      1  42727  42801      69.94   -3
3     1      1  68920  68995      78.62  -15
4     1      0 158662 158736      55.75  -17
5     1      0 158744 158827      52.31  -19
6     1      1 202339 202414      68.64  -10
... with 41 more rows.
> class(rnas)
[1] "Genes"
```

Wow! That has to be one of life's simplest pleasures. `FindNonCoding` returned an object of class *Genes*. By convention, the starting position of non-coding RNAs on the forward (`0`) `Strand` is at `Begin`, while those on the reverse (`1`) `Strand` start at `End`. We can take a look at which RNAs were found:

```
> annotations <- attr(rnas, "annotations")
> m <- match(rnas[, "Gene"], annotations)
> sort(table(names(annotations)[m]))
                   IhtA RNase_P_class_A-RF00010        SmallSRP-RF00169
                      1                         1                       1
               tRNA-Asn                  tRNA-Asp                tRNA-Cys
                      1                         1                       1
               tRNA-Gln                  tRNA-Glu                tRNA-His
                      1                         1                       1
               tRNA-Ile                  tRNA-Lys                tRNA-Phe
                      1                         1                       1
               tRNA-Trp                  tRNA-Tyr           tmRNA-RF00023
                      1                         1                       1
      rRNA_16S-RF00177         rRNA_23S-RF02541         rRNA_5S-RF00001
                      2                         2                       2
               tRNA-Ala                  tRNA-Gly                tRNA-Pro
                      2                         2                       2
               tRNA-Val                  tRNA-Arg                tRNA-Met
                      2                         3                       3
               tRNA-Thr                  tRNA-Ser                tRNA-Leu
                      3                         4                       5
```

We see that the *C. trachomatis* genome has multiple tRNA genes, two copies of each ribosomal RNA gene, and the RNaseP and tmRNA genes. Finally, it is possible to extract the non-coding RNAs from the genome:

```
> ExtractGenes(rnas, genome, type="RNAStringSet")
RNAStringSet object of length 47:
     width seq
 [1]   420 GGGGGUGUAAAGGUUUCGACUUAGAAAUGAAGC...AGGACGAGAGUUCGACUCUCUCCACCUCCAUAG
 [2]    75 UCCGGAGUAGCUCAGCGGUAGAGCAGUGGACUG...UGGUCGUUGGUUCGAACCCAUCCUCCGGAGUCU
 [3]    76 CGGAGUAUAGCGCAGCCUGGUUAGCGCGGUUGC...AUAGGUCGGGGGUUCGAAUCCCUCUACUCCGAU
 [4]    75 GCUGGAGUAGCUCAAUUGGCAGAGCAUUCGAUU...ACGGUUGAGGGUUCAAUUCCUUUCUCCAGCAUC
 [5]    84 GGGGGUGUCGCAUAGCGGUCAAUUGCAUCGGAC...CGGAUACGUUGGUUCAAAUCCAGCCACCCCCAG
 ...   ... ...
[43]    72 GGUGGCAUCGCCAAGCGGUAAGGCCGAGGCCUG...CUCUAUCCCCGGUUCGAUUCCGGGUGCCACCUU
[44]    74 UGGGGUGUGGCCAAGCGGUAAGGCAGCGGUUUU...CGCAUCGGAGGUUCGAAUCCUUCCACCCCAGAG
[45]    75 GGGGUAUUAGCUCAGUUGGUUAGAGCGUCACGU...GAAGGUCAGCUGUUCAAGUCAGCUAUAUCCCAA
[46]    88 GGAAGAAUGGCAGAGCGGUUUAAUGCACCUGUC...GGUCCGGGGGUUCGAAUCCCUCUUCUUCCGCAU
[47]    85 GCCCAGGUGGUGAAAUUGGUAGACACGCUGGAU...GGCAUGUAGGUUCAAGUCCUAUCCUGGGCAUAG
```

You'll be the life of the party now that you know how to build models for non-coding RNAs and find them in a genome.

5 Session Information

All of the output in this vignette was produced under the following conditions:

- R version 4.2.0 RC (2022-04-19 r82224), `x86_64-apple-darwin17.0`
- Running under: `macOS Mojave 10.14.6`
- Matrix products: default

- BLAS:
  `/Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib`
- LAPACK:
  `/Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.42.0, Biostrings 2.64.0, DECIPHER 2.24.0, GenomeInfoDb 1.32.0, IRanges 2.30.0, RSQLite 2.2.12, S4Vectors 0.34.0, XVector 0.36.0
- Loaded via a namespace (and not attached): DBI 1.1.2, GenomeInfoDbData 1.2.8, KernSmooth 2.23-20, RCurl 1.98-1.6, Rcpp 1.0.8.3, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.3, cachem 1.0.6, cli 3.3.0, compiler 4.2.0, crayon 1.5.1, fastmap 1.1.0, memoise 2.0.1, pkgconfig 2.0.3, rlang 1.0.2, tools 4.2.0, vctrs 0.4.1, zlibbioc 1.42.0