

The Double Life of RNA: Uncovering Non-Coding RNAs

Erik S. Wright

August 19, 2026

Contents

1	Introduction	1
2	Getting Started	1
2.1	Startup	1
3	Building a Non-Coding RNA Model	2
3.1	Importing the sequences	2
3.2	Aligning the sequences	3
3.3	Learning sequence patterns	8
4	Finding Non-Coding RNAs	9
5	Session Information	13

1 Introduction

RNA leads a double life as both messenger (mRNA) and number of other 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 the sequence of RNA folds into a stable structure, can be found for sequences in many regions of the genome. The ubiquity of stable secondary structures makes life difficult because there is no clear signature that a genomic region contains a non-coding RNA in the way there are distinct signatures for protein coding regions. The most promising approach is to search for intergenic regions with high GC-content and 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 lhtA, that has only been found in Chlamydiae and is believed to play a role in regulating its life stages. We are going to find lhtA 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	AAGAUGAUUUCUGCGCCAUGGA...UGAUUGUUGUUUUCUUGGCUUU	KE360988.1/6443-6...
[2]	105	AAGAUGAUUUCUCCGCCGUGGA...AACUUUAUGUUUUCUUGGCUUU	AE002161.1/54676-...
[3]	107	AAGUUGGUAUUCUAACGCCAUGG...UAUCUCCGGUUCUCUUGGCUUU	AE001273.1/773281...
[4]	107	AAGUUGGUAUUCUAACGCCAUGG...UGUCUCCAGUUCUCUUGGCUUU	AE002160.2/53054-...
[5]	103	AAGAUGAUUUCUACGCCAUGGA...UGAUUGUUGUUUUCUUGGCUUU	AE015925.1/52097-...
...	...	...	...
[23]	107	AAGUUGGUAUUCUAACGCCAUGG...UAUCUCCGGUUCUCUUGGCUUU	CP006674.1 Chlamy...
[24]	103	AAGAUGAUUUCUGCGCCAUGGA...UGAUUGUUGUUUUCUUGGCUUU	KE356008.1 Chlamy...
[25]	108	AAGUUGGUAUUCUAACGCCAUGG...UAUCUCCGGUUCUCUUGGCUUU	CVNC01000001.1 Ch...
[26]	103	AAGAUGAUUUCUGCGCCAUGGA...UGAUUGUUGUUUUCUUGGCUUU	AP006861.1 Chlamy...
[27]	101	AAGAUGAUUUCUACGCCAUGGA...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)
Predicting structures based on free energies:
=====

Time difference of 0.36 secs

Determining distance matrix based on shared 7-mers:
=====

Time difference of 0 secs

Clustering into groups by similarity:
=====

Time difference of 0 secs

Aligning Sequences:
=====

Time difference of 0.06 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 secs

Realigning Sequences:
=====

Time difference of 0.04 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 secs

Realigning Sequences:

=====

Time difference of 0.01 secs

> RNA

RNAStringSet object of length 27:

	width	seq	names
[1]	114	AAGAUGAUAUUCUG-CGCCAUGG...GA-UUGUUGUUUUCUUGGCUUU	KE360988.1/6443-6...
[2]	114	AAGAUGAUAUUCUC-CGCCGUGG...AACUUUAUGUUUUCUUGGCUUU	AE002161.1/54676-...
[3]	114	AAGUUGGUUAUUCUAACGCCAUGG...---UCCGGUUCUCUUGGCUUU	AE001273.1/773281...
[4]	114	AAGUUGGUUAUUCUAACGCCAUGG...---UCCAGUUCUCUUGGCUUU	AE002160.2/53054-...
[5]	114	AAGAUGAUAUUCUA-CGCCAUGG...GA-UUGUUGUUUUCUUGGCUUU	AE015925.1/52097-...
...	...	...	...
[23]	114	AAGUUGGUUAUUCUAACGCCAUGG...---UCCGGUUCUCUUGGCUUU	CP006674.1 Chlamy...
[24]	114	AAGAUGAUAUUCUG-CGCCAUGG...GA-UUGUUGUUUUCUUGGCUUU	KE356008.1 Chlamy...
[25]	114	AAGUUGGUUAUUCUAACGCCAUGG...---UCCGGUUCUCUUGGCUUU	CVNC01000001.1 Ch...
[26]	114	AAGAUGAUAUUCUG-CGCCAUGG...GA-UUGUUGUUUUCUUGGCUUU	AP006861.1 Chlamy...
[27]	114	AAGAUGAUAUUCUA-CGCCAUGG...GA--UGUUGUUUUCUUGGCUUU	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 (Predict Dot-Bracket Notation) 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 secs

Computing Free Energies:

=====

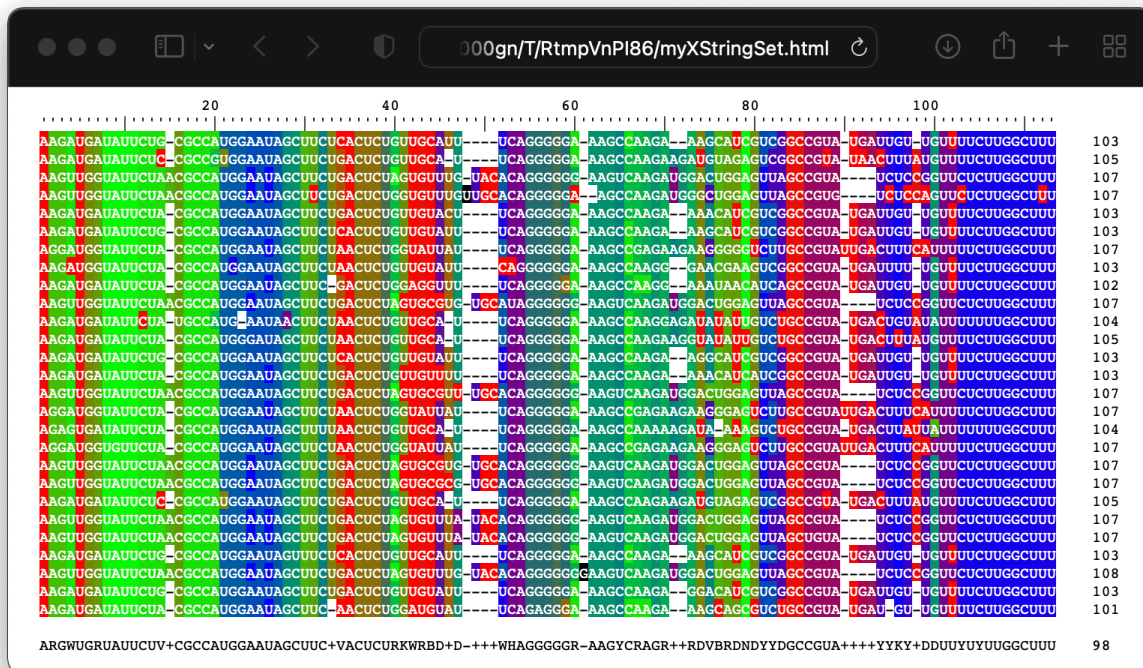
Time difference of 0.18 secs

Predicting RNA Secondary Structures:

=====

Time difference of 0 secs

> BrowseSeqs(RNA, patterns=p)



We can also visualize the secondary structure through a dot plot (Fig. 2). The top half of the dot plot shows the probabilities of pairing, and the bottom half shows the pairs selected for 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)

```

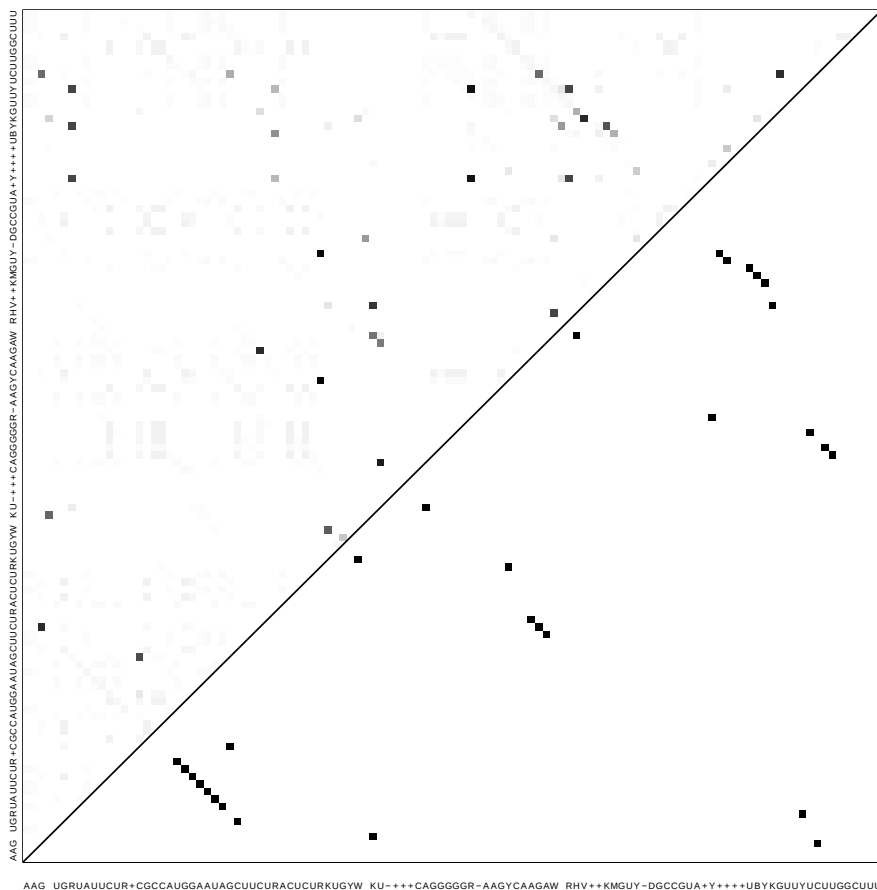


Figure 2: Secondary Structure Dot Plot.

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, 5 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. Each row stores information about where the motif is expected relative to the ncRNA, as well as its position weight matrix (pwm) and associated scores.

```
> y[["motifs"]]
      begin_low begin_high end_low end_high      motif      pwm
1           0           0      96      100      AAGATGr 0.946800....
2           7           7      89       93      TATTCTA 0.017733....
3          14          15      82       86      CGCCATG 0.018053....
4          21          22      81       85           G 0.017733....
5          22          23      72       76      AATAGCTTC 0.908696....
6          31          33      59       63 rACTCTGkTGyw 0.446377....
7          44          46      56       60           TT 0.041726....
8          46          51      48       52      CAGGGGGA 0.085348....
9          54          59      47       51           A 0.946800....
10         55          60      34       38 AGCCAAGAarrhr 0.946800....
11         69          75      27       31          kmGTC 0.221530....
12         74          80      20       24      dGCCGTA 0.275894....
13         82          86      19       20           A 0.946800....
14         84          88      18       18           T 0.017733....
15         85          89       8        8      TkytGTTTTTC 0.018053....
16        95          99       0         0      TTGGCTTT 0.017733....

      minscore prevalence background
1 0, 2.082.... 0.035714.... 0.970011....
2 0, 5.871.... 0.035714.... 0.977774....
3 0, 6.119.... 0.035714.... 0.983434....
4      0, Inf 0.057263.... 0.550452....
5 0, 8.018.... 0.057263.... 0.985521....
6 0, 6.540.... 0.035714.... 0.976631....
```

```

7 0.756512.... 0.123059.... 0.708078....
8 5.040460.... 0.131007.... 0.997916....
9      0, Inf 0.035714.... 0.258942....
10 0, 9.459.... 0.061810.... 0.981447....
11 2.460447.... 0.206663.... 0.904872....
12 4.220658.... 0.021254.... 0.989102....
13      0, Inf 0.112712.... 0.582480....
14      0, Inf 0.281062.... 0.670569....
15 0, 3.463.... 0.035714.... 0.962619....
16      0, Inf 0.035714.... 0.990812....

```

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      52      58         46         57          6
2          7          7      77      81         19         20          8
3         23         29      35      40         39         46          5
4         34         38      40      44         22         29          5
5         49         54      -4      15         38         58          7
  length_high      dG prevalence background
1         15 -Inf, -4.... 0.107097.... 0.090727....
2          8 -Inf, -3.... 0.743057.... 0.008820....
3          9 -Inf, -4.... 0.255296.... 0.088102....
4          9 -Inf, -2.... 0.195755.... 0.112149....
5         26 -Inf, -2.... 0.139890.... 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] 184 109 183 162  78  84
> tail(y[["kmers"]])
[1] 236 222 216 218 155 339

```

Finally, the sequences are identifiable by their k-mer frequencies, which are stored in a vector as shown above. 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 GCGGCCGCCCCGGGAAATTGCTA...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 3.95 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      94.94  -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 rows are predicted non-coding RNAs and not protein coding genes. In this manner, non-coding and coding genes can be combined in the same *Genes* object.

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 28.16 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	88.31	-25
2	1	1	42727	42801	69.82	-3
3	1	1	68920	68995	78.58	-15
4	1	0	158662	158736	55.72	-17
5	1	0	158744	158827	52.24	-19
6	1	1	202339	202414	68.58	-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
tmRNA-RF00023	tRNA-Asn	tRNA-Asp
1	1	1
tRNA-Cys	tRNA-Gln	tRNA-Glu
1	1	1
tRNA-His	tRNA-Ile	tRNA-Lys
1	1	1
tRNA-Phe	tRNA-Trp	tRNA-Tyr
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...AUAGGUCGGGGUUCGAAUCCCUACUCCGAU
[4]    75 GCUGGAGUAGCUCAAUUGGCAGAGCAUUCGAUU...ACGGUUGAGGGUUCAAUCCCUUUCUCCAGCAUC
[5]    84 GGGGGUGUCGCAUAGCGGUCAAUUGCAUCGGAC...CGGAUACGUUGGUUCAAUCCAGCCACCCCCAG
...
[43]    72 GGUGGCAUCGCCAAGCGGUAAGGCCGAGGCCUG...CUCUAUCCCCGGUUCGAUUCGGGUGCCACCUU
[44]    74 UGGGGUGUGGCCAAGCGGUAAGGCAGCGGUUUU...CGCAUCGGAGGUUCGAAUCCUCCACCCCAGAG
[45]    75 GGGGUAUUAGCUCAGUUGGUUAGAGCGUCACGU...GAAGGUCAGCUGUUAAGUCAGCUAUAUCCCAA
[46]    88 GGAAGAAUGGCAGAGCGGUUUAUUGCACCUGUC...GGUCCGGGGUUCGAAUCCCUUUCUCCGCAU
[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 4.6.1 (2026-06-24), x86_64-pc-linux-gnu
- Running under: Ubuntu 26.04 LTS
- Matrix products: default
- BLAS: /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
- LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblas-p-r0.3.32.so
; LAPACK version 3.12.0
- Base packages: base, datasets, graphics, grDevices, methods, stats, stats4, utils
- Other packages: BiocGenerics 0.59.12, Biostrings 2.81.6, DECIPHER 3.9.3, generics 0.1.4, IRanges 2.47.2, S4Vectors 0.51.6, Seqinfo 1.3.0, XVector 0.53.0
- Loaded via a namespace (and not attached): buildtools 1.0.0, cli 3.6.6, compiler 4.6.1, crayon 1.5.3, DBI 1.3.0, evaluate 1.0.5, KernSmooth 2.23-26, knitr 1.51, maketools 1.3.2, rlang 1.3.0, sys 3.4.3, tools 4.6.1, xfun 0.60