The Magic of Gene Finding

Erik S. Wright

October 29, 2020

Contents

1 Introduction

This vignette reveals the tricks behind the magic that is *ab initio* gene finding. Cells all have the magical ability to transcribe and translate portions of their genome, somehow decoding key signals from a sea of possibilities. The `FindGenes` function attempts to decipher these signals in order to accurately predict an organism's set of genes. Cells do much of this magic using only information upstream of the gene, whereas `FindGenes` uses both the content of the gene and its upstream information to predict gene boundaries. As a case study, this tutorial focuses on finding genes 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. Nevertheless, `FindGenes` is designed to work with any genome that lacks introns, making it well-suited for prokaryotic gene finding.

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)
```

Gene finding is performed with the function `FindGenes`. Help can be accessed via:

```
> ? FindGenes
```

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

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

3 Finding Genes in a Genome

Ab initio gene finding begins from a genome and locates genes without prior knowledge about the specific organism.

3.1 Importing the genome

The first step is to set filepaths to the genome sequence (in FASTA format). 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:
> 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...
```

3.2 Finding genes

The next step is to find genes in the genome using `FindGenes`, which does all the magic. There are fairly few choices to make at this step. By default, the bacterial and archaeal genetic code is used for translation, including the initiation codons "ATG", "GTG", "TTG", "CTG", "ATA", "ATT", and "ATC". The default *minGeneLength* is `60`, although we could set this lower (e.g., `30`) to locate very short genes or higher (e.g., `90`) for (only slightly) better accuracy. The argument *allowEdges* (default `TRUE`) controls whether genes are allowed to run off the ends of the genome, as would be expected for circular or incomplete chromosomes. Here, we will only set *showPlot* to `TRUE` to display a summary of the gene finding process and *allScores* to `TRUE` to see the scores of all open reading frames (including predicted genes).

```
> orfs <- FindGenes(genome, showPlot=TRUE, allScores=TRUE)
Iter  Models Start Motif  Fold  Init UpsNt  Term   RBS  Auto  Stop Genes
1          1  7.50                                                  921
2          1 15.00  0.44  0.38  0.96                                910
3          1 15.14  0.53  0.52  1.22  1.33                          904
4          2  9.76  0.53  0.54  1.29  1.70  0.24                    906
5          2  9.81  0.53  0.54  1.25  1.72  0.24  0.93  0.10  0.07  907
6          2  9.76  0.51  0.54  1.23  1.85  0.24  1.01  0.10  0.07  908
7          2  9.74  0.51  0.53  1.23  1.87  0.25  1.01  0.10  0.07  910
8          2  9.73  0.52  0.52  1.23  1.91  0.25  1.04  0.10  0.07  909
9          2  9.72  0.52  0.52  1.24  1.93  0.25  1.07  0.10  0.08  909
10         2  9.72  0.52  0.52  1.24  1.93  0.25  1.06  0.10  0.08  909

Time difference of 78.16 secs
```

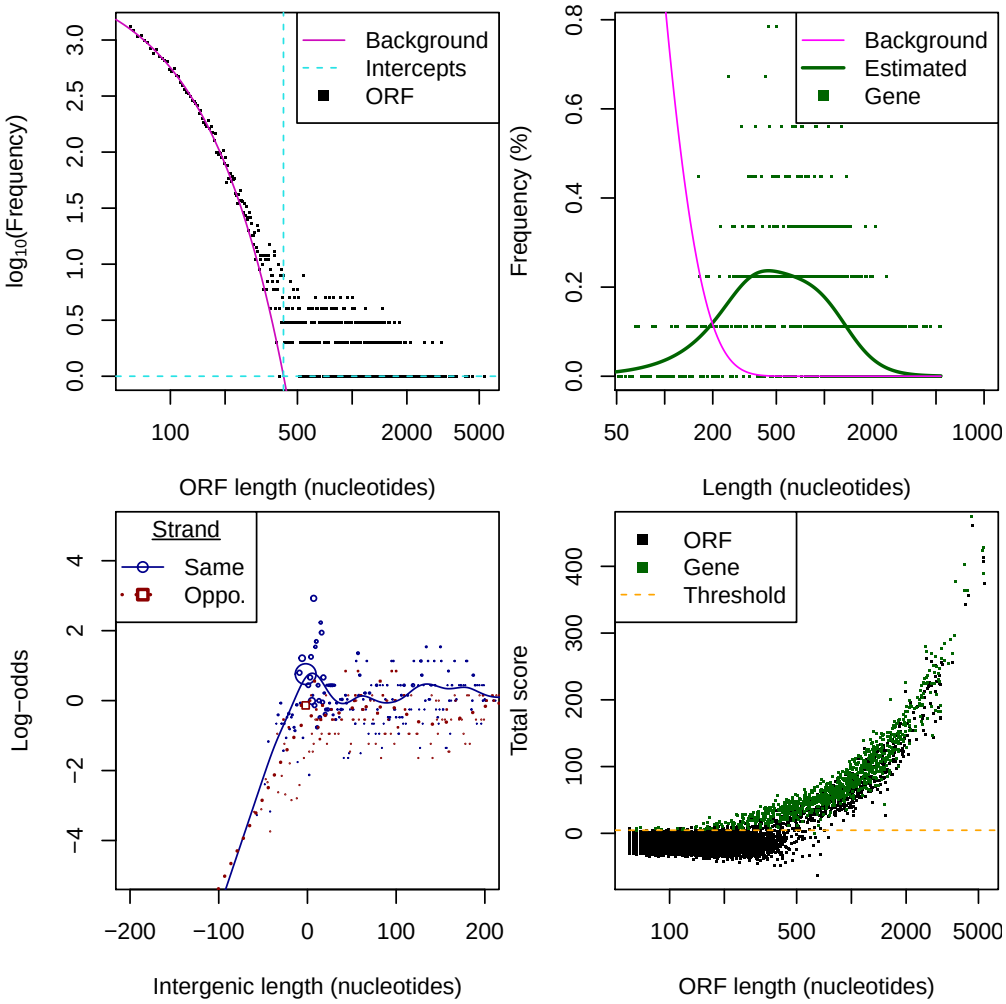

Figure 1: Plot summarizing the gene finding process of `FindGenes`. See `?plot.Genes` for details.

3.3 Inspecting the output

And presto! We now have our gene predictions in the form of an object belonging to class *Genes*. Now that we have our genes in hand, let's take a look at them:

```
> orfs
Genes object specifying 909 genes from 66 to 5,361 nucleotides.
Density: 90.2%; Initation codons: 4; Score correlations: -22% to 53%.

  Index Strand Begin  End TotalScore ... FractionReps Gene
1     1      0     1 1176      75.23 ...            1    1
2     1      0     2   82      -5.83 ...            0    0
3     1      0    16 1176      67.38 ...            0    0
4     1      0    17   82      -9.62 ...            0    0
5     1      1    41  106     -12.80 ...            0    0
6     1      1    91  150     -25.33 ...            0    0
... with 109,484 more rows.
```

Here, we can see that the genome has a wide range of gene lengths, high density of predicted genes, uses multiple initiation codons, and contains many possible open reading frames. We see all the open reading frames in the output because *allScores* was set to `TRUE`. If we only want to look at the subset of open reading frames that are predicted as genes, we can subset the object:

```
> genes <- orfs[orfs[, "Gene"]==1,]
```

The `"Gene"` column is one of several describing the open reading frames. Objects of class *Genes* are stored as matrices with many columns containing information about the open reading frames:

```
> colnames(genes)
 [1] "Index"                   "Strand"
 [3] "Begin"                   "End"
 [5] "TotalScore"              "LengthScore"
 [7] "CodingScore"             "StartScore"
 [9] "StopScore"               "InitialCodonScore"
[11] "TerminationCodonScore"   "RibosomeBindingSiteScore"
[13] "AutocorrelationScore"    "UpstreamNucleotideScore"
[15] "UpstreamMotifScore"      "FoldingScore"
[17] "FractionReps"            "Gene"
```

4 Analyzing the Output

4.1 Extracting genes from the genome

Predictions in hand, the first thing we can do is extract the genes from the genome. This can be easily done using `ExtractGenes`.

```
> dna <- ExtractGenes(genes, genome)
> dna
DNAStringSet object of length 909:
      width seq                                               names
  [1]  1176 GCGGCCGCCCGGGAAATTGCTA...TAGAGAAGTCGATAGGGAATAA NC_000117.1 Chlam...
```

```
  [2]   273 ATGCTTTGTAAAGTTTGTAGAG...CCATCATATGGATAGAGAATAA NC_000117.1 Chlam...
  [3]   303 ATGACAGAGTCATATGTAAACA...AGTCCCTACAGTTATCAAATAG NC_000117.1 Chlam...
  [4]  1476 ATGTATCGTAAGAGTGCTTTAG...TTTTGACGGAGGAATAGAATAA NC_000117.1 Chlam...
  [5]  1467 ATGGGCATAGCACATACTGAAT...AGCAGCTATGCGAGATATGTAA NC_000117.1 Chlam...
  ...   ... ...
[905]  3051 ATGCCTTTTTCTTTGAGATCTA...GGGCTTGAATAGAATCTTTTAA NC_000117.1 Chlam...
[906]    90 ATGATTCACCTCGATTCCGTTA...AGGGCGTATTTTTTATTATTAA NC_000117.1 Chlam...
[907]   303 ATGCTGGCAACAATTAAAAAAA...CCTAGAATGCAAGAAGATATAA NC_000117.1 Chlam...
[908]  2637 ATGCGACCTGATCATATGAACT...GGGGACCACTTACAGGTTCTAG NC_000117.1 Chlam...
[909]   600 ATGAGCATCAGGGGAGTAGGAG...GTTGGCTGGCCCTGACGGGGTA NC_000117.1 Chlam...
```

We see that the first gene has no start codon and the last gene has no stop codon. This implies that the genes likely connect to each other because the genome is circular and the genome end splits one gene into two. Therefore, the first predicted gene's first codon is not a true start codon and we need to drop this first sequence from our analysis of start codons. We can look at the distribution of predicted start codons with:

```
> table(subseq(dna[-1], 1, 3))
ATG CTG GTG TTG
795   1  88  24
```

There is one predicted non-canonical initiation codon, "CTG", and the typical three bacterial initiation codons: "ATG", "GTG", and "TTG". Let's take a closer look at genes with non-canonical initiation codons:

```
> w <- which(!subseq(dna, 1, 3) %in% c("ATG", "GTG", "TTG"))
> w
[1]   1 606
> w <- w[-1] # drop the first sequence because it is a fragment
> w
[1] 606
> dna[w]
DNAStringSet object of length 1:
    width seq                                                 names
[1]  1992 CTGAAACAGCAATTTGTATTACA...GAGAAGAAGAAAATGGGGACTGA NC_000117.1 Chlam...
> genes[w,]
Genes object specifying 1 gene of 1,992 nucleotides.
Density: 0.2%; Initation codons: 1.

      Index Strand  Begin    End TotalScore ... FractionReps Gene
69039     1      0 659697 661688     175.39 ...            1    1
```

We can also look at the predicted protein sequences by translating the genes:

```
> aa <- ExtractGenes(genes, genome, type="AAStringSet")
> aa
AAStringSet object of length 909:
      width seq                                               names
  [1]   392 AAAREIAKRWEQRVRDLQDKGA...TNGSQTFRDLMRRWNREVDRE* NC_000117.1 Chlam...
  [2]    91 MLCKVCRGLSSLIVVLGAINTG...KRHGDCCSSKGGYHHHHMDRE* NC_000117.1 Chlam...
  [3]   101 MTESYVNKEEIISLAKNAALEL...EFLSNVPVSLGGLVKVPTVIK* NC_000117.1 Chlam...
  [4]   492 MYRKSALELRDAVVNRELSVTA...HSQIKQLYPKAVNGLFDGGIE* NC_000117.1 Chlam...
```

```
  [5]   489 MGIAHTEWESVIGLEVHVELNT...TEGKAPPKRVNELLLAAMRDM* NC_000117.1 Chlam...
  ...   ... ...
[905]  1017 MPFSLRSTSFCFLACLCSYSYG...SLDARRRQTAHFVSMGLNRIF* NC_000117.1 Chlam...
[906]    30 MIHLDSVRKQNGFNKNKAVLAELGRIFYY*                  NC_000117.1 Chlam...
[907]   101 MLATIKKITVLLLSKRKAGIRI...RLAGLMLDYWDGDSRLECKKI* NC_000117.1 Chlam...
[908]   879 MRPDHMNFCCLCAAILSSTAVL...SYVLRGQSHSYSLDLGTTYRF* NC_000117.1 Chlam...
[909]   200 MSIRGVGGNGNSRIPSHNGDGS...AAYTSECADHLEANKLAGPDGV NC_000117.1 Chlam...
```

All of the genes start with a methionine ("M") residue and end with a stop ("*") except the first and last gene because they wrap around the end of the genome. If so inclined, we could easily remove the first and last positions with:

```
> subseq(aa, 2, -2)
AAStringSet object of length 909:
      width seq                                               names
  [1]   390 AAREIAKRWEQRVRDLQDKGAA...LTNGSQTFRDLMRRWNREVDRE NC_000117.1 Chlam...
  [2]    89 LCKVCRGLSSLIVVLGAINTGI...KKRHGDCCSSKGGYHHHHMDRE NC_000117.1 Chlam...
  [3]    99 TESYVNKEEIISLAKNAALELE...EEFLSNVPVSLGGLVKVPTVIK NC_000117.1 Chlam...
  [4]   490 YRKSALELRDAVVNRELSVTAI...EHSQIKQLYPKAVNGLFDGGIE NC_000117.1 Chlam...
  [5]   487 GIAHTEWESVIGLEVHVELNTE...RTEGKAPPKRVNELLLAAMRDM NC_000117.1 Chlam...
  ...   ... ...
[905]  1015 PFSLRSTSFCFLACLCSYSYGF...YSLDARRRQTAHFVSMGLNRIF NC_000117.1 Chlam...
[906]    28 IHLDSVRKQNGFNKNKAVLAELGRIFYY                    NC_000117.1 Chlam...
[907]    99 LATIKKITVLLLSKRKAGIRID...VRLAGLMLDYWDGDSRLECKKI NC_000117.1 Chlam...
[908]   877 RPDHMNFCCLCAAILSSTAVLF...VSYVLRGQSHSYSLDLGTTYRF NC_000117.1 Chlam...
[909]   198 SIRGVGGNGNSRIPSHNGDGSN...MAAYTSECADHLEANKLAGPDG NC_000117.1 Chlam...
```

4.2 Revealing the secrets of gene finding

The predictions made by `FindGenes` are supported by many scores that are mostly independent of each other. However, some scores are related because they make use of information from the same region relative to the gene boundaries. We can take a look at score correlations with:

```
> pairs(genes[, 5:16], pch=46, col="#00000033", panel=panel.smooth)
```

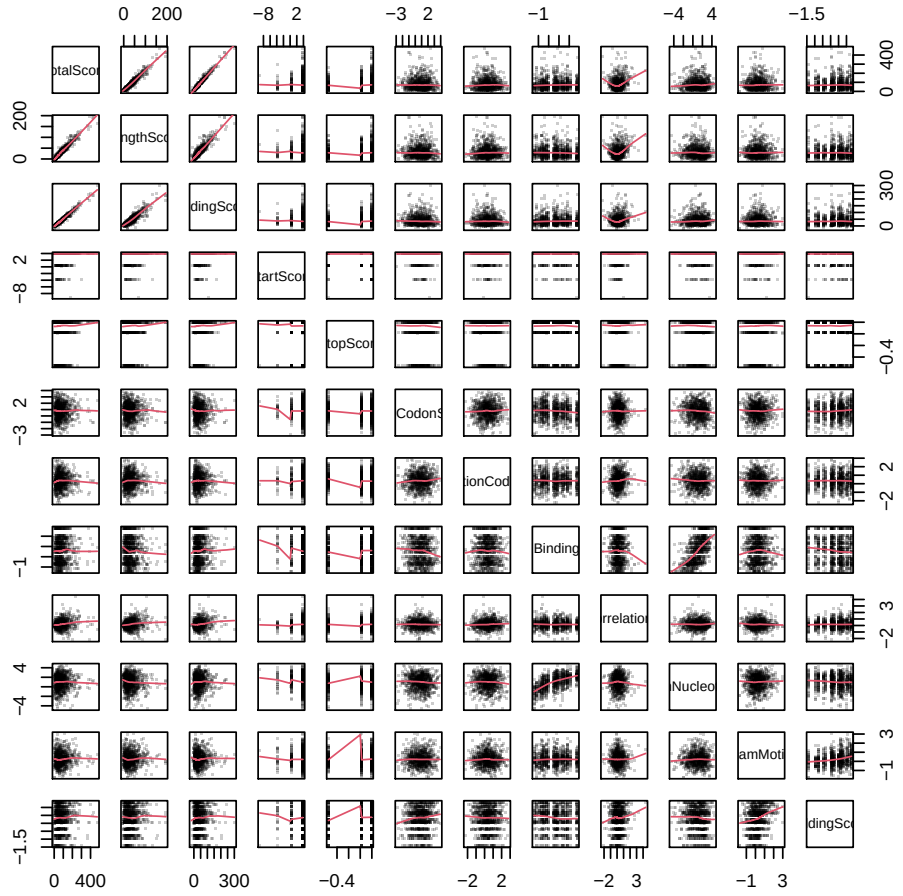

Figure 2: Scatterplot matrix of scores used by `FindGenes` to make gene predictions.

Certainly some of the magic of gene finding is having a lot of scores! We see that the ribosome binding site score and upstream nucleotide score are the most correlated, which is unsurprising because they both rely on the nucleotides immediately upstream of the start codon. The different scores are defined as follows:

1. Upstream signals
   (a) *Ribosome Binding Site Score* - Binding strength and position of the Shine-Delgarno sequence, as well as other motifs in the first bases upstream of the start codon.
   (b) *Upstream Nucleotide Score* - Nucleotides in each position immediately upstream of the start codon.
   (c) *Upstream Motif Score* - K-mer motifs further upstream of the start codon.
2. Start site signals
   (a) *Start Score* - Choice of start codon relative to the background distribution of open reading frames.
   (b) *Folding Score* - Free energy of RNA-RNA folding around the start codon and relative to locations upstream and downstream of the start.
   (c) *Initial Codon Score* - Choice of codons in the first few positions after the start codon.
3. Gene content signals
   (a) *Coding Score* - Usage of codons or pairs of codons within the open reading frame.
   (b) *Length Score* - Length of the open reading frame relative to the background of lengths expected by chance.
   (c) *Autocorrelation Score* - The degree to which the same or different codons are used sequentially to code for an amino acid.
4. Termination signals
   (a) *Termination Codon Score* - Codon bias immediately before the stop codon.
   (b) *Stop Score* - Choice of stop codon relative to the observed distribution of possible stop codons.

4.3 Taking a closer look at the output

If we have a particular gene of interest, it can sometimes be useful to plot the output of `FindGenes` as the set of all possible open reading frames with the predicted genes highlighted. The `plot` function for a *Genes* object is interactive, so it is possible to pan left and right by setting the *interact* argument equal to `TRUE`. For now we will only look at the beginning of the genome:

```
> plot(orfs, interact=FALSE)
```

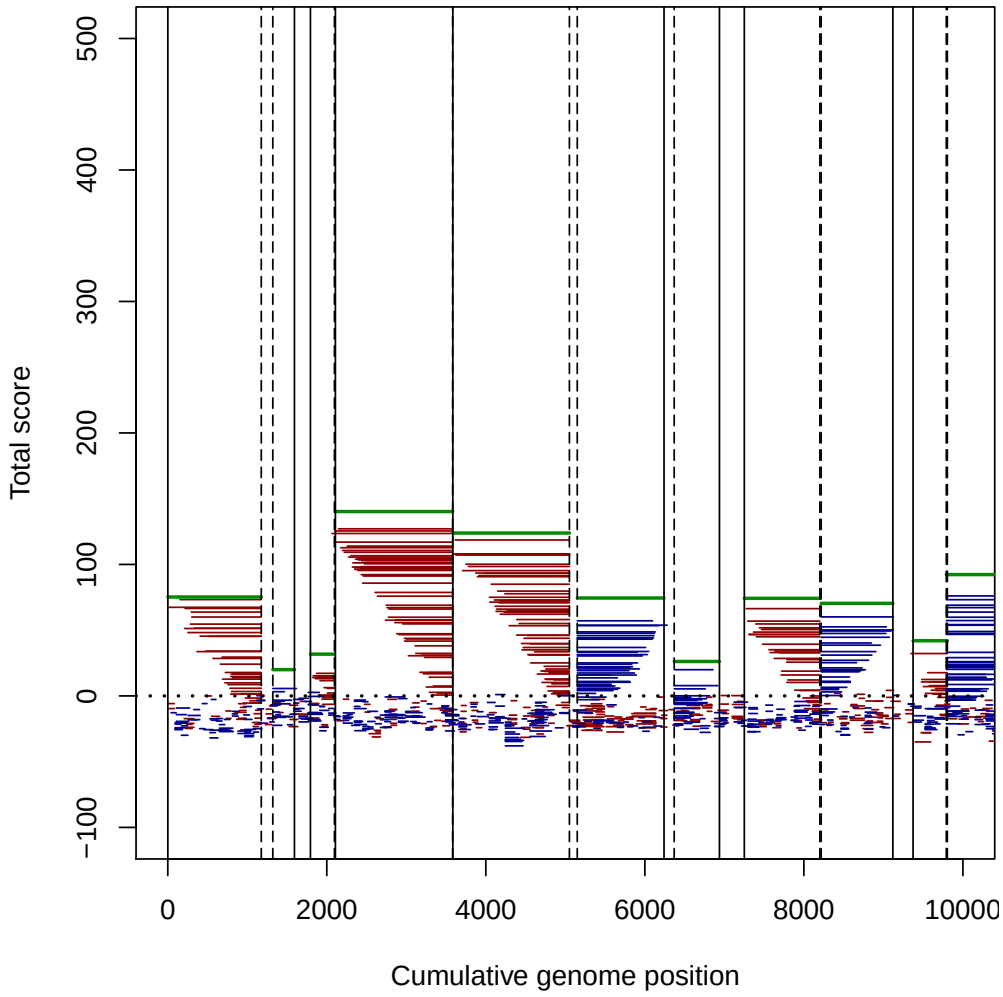

Figure 3: All possible open reading frames (red and blue) with predicted genes highlighted in green.

5 Exporting the output

The genes can be exported in a variety of formats, including as a FASTA file with `writeXStringSet`, GenBank (gbk) or general feature format (gff) file with `WriteGenes`, or delimited file formats (e.g., csv, tab, etc.) with `write.table`.

Now that you know the tricks of the trade, you can work your own magic to find new genes!

6 Session Information

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

- R version 4.0.3 (2020-10-10), `x86_64-apple-darwin17.0`
- Running under: `macOS Mojave 10.14.6`
- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib`
- LAPACK: `/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.36.0, Biostrings 2.58.0, DECIPHER 2.18.1, IRanges 2.24.0, RSQLite 2.2.1, S4Vectors 0.28.0, XVector 0.30.0
- Loaded via a namespace (and not attached): DBI 1.1.0, KernSmooth 2.23-18, Rcpp 1.0.5, bit 4.0.4, bit64 4.0.5, blob 1.2.1, compiler 4.0.3, crayon 1.3.4, digest 0.6.27, memoise 1.1.0, pkgconfig 2.0.3, rlang 0.4.8, tools 4.0.3, vctrs 0.3.4, zlibbioc 1.36.0