Predicting maximal growth rates with gRodon

The gRodon package allows you to predict the maximal growth rate of a prokaryotic organism from genomic data. For details of how this is done please see the original paper. Briefly, gRodon exploits codon usage statistics to detect optimization of highly-expressed genes, an indicator of selection for rapid growth. We will provide some guidance about when these predictions are appropriate below.

The gRodon package is quite simple, with only a single function currently available to users: `predictGrowth()`. We provide a simple, easy to use interface that allows any user with a genome in-hand to predict maximal growth rate.

The gRodon approach is largely based on an earlier program, `growthpred`, developed by Vieira-Silva et al.. gRodon is intended to be:
1. User-friendly (especially for R users)
2. More accurate (due to the incorporation of additional measures of codon usage)
3. More accurate on mixed-organism samples (i.e., metagenomes; due to a correction factor for the relative abundance of different organisms)

Installation

The easiest way to install gRodon is with devtools:

```
devtools::install_github("jlw-ecoevo/gRodon2")
```

gRodon has a few dependencies - namely the `Biostrings`, `coRdon`, and `matrixStats` packages which are bioconductor packages and cannot be installed via CRAN. To install them run the following:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("Biostrings")
BiocManager::install("coRdon")
install.packages("matrixStats")
```

Before You Begin

There are two things that gRodon does *not* do that you *must* do ahead of time yourself in order to predict growth rate:
1. You must identify coding sequences in your genome
(e.g., using prodigal)
2. You must annotate a set of highly-expressed genes, typically those coding for ribosomal proteins. We usually use genes coding for ribosomal proteins as our highly-expressed set since these are universal across microbes and almost always highly expressed (if you wanna grow you gotta make proteins), but if you have expression data for your organism you might be able to tailor an even better set of highly expressed genes.

It is helpful if (in frame) coding sequences are stored in a fasta file for easy loading. Additionally, you will need to have a logical vector describing which genes are part of your highly expressed set (this should become a bit clearer with the examples below).

A Tiny Detour: Annotating Your Assembly I'll lead you through one way to get annotated coding sequence to input into gRodon here, but there are many (MANY) ways to do this. For example, check out

this set of tutorials on functional annotation.

Disclaimer: *I make no claim that this is the best, most efficient, or most graceful way to get annotated coding sequences from your assembly, but it will get the job done. If you have strong opinions about how this should be done shoot me an email and I will be happy to link to your rant/tutorial. If you already have annotated coding sequences please skip this section. If you have unannotated CDS (e.g., from prodigal), you might consider simply BLASTing your output for highly expressed genes (e.g., using the blastdb of ribosomal proteins included in growthpred).*

Given an un-annotated genome or metagenomic assembly, an easy way to find and annotate coding sequence is using the program prokka. For example, if I have a set of assembled contigs, I could run (on the command line):

```
prokka contig.fasta --prefix my_genome
```

This will give several output files. You are interested the untranslated coding sequences (CDS), which can be found in the `*.ffn` file along with other non-coding transcripts (e.g., rRNA, tRNA). To get the names of just your CDS you can pull them out of the `*.gff` output:

```
sed -n '/##FASTA/q;p' my_genome/my_genome.gff | awk '$3=="CDS"' | awk '{print $9'} | awk 'gsub(";.*","")
```

The above command first trims the sequence data off the end of the `*.gff` file, then looks for rows describing CDS and pulls out their IDs, finally it outputs these IDs to a file called `CDS_names.txt`.

```
# Load your *.ffn file into R
genes <- readDNAStringSet("my_genome/my_genome.ffn")

# Subset your sequences to those that code for proteins
CDS_IDs <- readLines("CDS_names.txt")
gene_IDs <- gsub(" .*","",names(genes)) #Just look at first part of name before the space
genes <- genes[gene_IDs %in% CDS_IDs]

#Search for genes annotated as ribosomal proteins
highly_expressed <- grepl("^(?!.*(methyl|hydroxy)).*0S ribosomal protein",names(genes),ignore.case = T,
```

You are now ready to run gRodon.

A Minimal Example

We have included an example genome assembly with this package, downloaded from NCBI's RefSeq database (*Streptococcus pyogenes* M1, GCF_000349925.2). This is simply a fasta file with the predicted coding sequences and annotations for this genome (provided by NCBI).

Let's load this file using the `readDNAStringSet()` function from the `Biostrings` package (required for `gRodon` to work).

```
library(gRodon)
library(Biostrings)
```

```
path_to_genome <- system.file('extdata',
  'GCF_000349925.2_ASM34992v2_cds_from_genomic.fna.gz',
  package = 'gRodon')
genes <- readDNAStringSet(path_to_genome)
```

We also need a set of highly expressed genes. In general, the ribosomal proteins are a good set of genes to use for this purpose. Since these proteins should already be annotated in our example file (try running `names(genes)` to see the annotations from NCBI) we can use `grep()` to search for them (specifically, `grepl()` to return a logical vector).

```
highly_expressed <- grepl("ribosomal protein",names(genes),ignore.case = T)
```

And now we are ready to predict the maximal growth rate of *S. pyogenes* M1.

```
predictGrowth(genes, highly_expressed)
#> Warning in filterSeq(genes = genes, highly_expressed = highly_expressed, :
#> There were 146 genes either with lengths not multiples of 3 or not above length
#> threshold (default 240bp), these genes have been ignored
#> $CUBHE
#> [1] 0.9148924
#>
#> $GC
#> [1] 0.3917167
#>
#> $GCdiv
#> [1] 0.1082833
#>
#> $ConsistencyHE
#> [1] 0.5062119
#>
#> $CUB
#> [1] 0.564834
#>
#> $CPB
#> [1] -0.3886505
#>
#> $FilteredSequences
#> [1] 146
#>
#> $nHE
#> [1] 45
#>
#> $dCUB
#> [1] -0.6197544
#>
#> $d
#> [1] 0.364274
#>
#> $LowerCI
#> [1] 0.2976978
#>
#> $UpperCI
#> [1] 0.4483901
```

The warning at the top is letting you know that there were a number of genes in your fasta that either weren't a length multiple of three, or were under the length threshold of 80 codons (240 bp) where estimates of bias will be unreliable. Don't worry about this unless the number of filtered genes is very high compared to the total number of genes in your file. We expect some genes to typically get filtered.

The output contains several (hopefully) useful quantities:

- `$CUBHE` is the mean of the codon usage bias of each highly expressed gene relative to all other genes
- `$ConsistencyHE` is the mean of the codon usage bias of each highly expressed gene relative to all other highly expressed genes
- `$CPB` is the genome-wide codon pair bias

- `$FilteredSequences` is the number of sequences filtered due to length
- `$d` is the estimated minimal doubling time from `gRodon` in hours. This is, presumably, what you came here for.
- `$LowerCI` and `$UpperCI` are the 95% confidence intervals for `d`

For details on how each of these is calculated please see the `gRodon` paper. For most users the only outputs that will matter much to you are `$d`, `$LowerCI`, and `$UpperCI`, which are your estimated minimal doubling times and 95% CIs output by the gRodon model.

For the most part, that's all you need to know. There are a few specific use cases we discuss below that require a slightly more thoughful application of `gRodon`. These are:

- The use of partial genomic data (e.g., incomplete SAGs or MAGs)
- The use of metagenomic data (**if using metagenomes you MUST use metagenome mode (v1 or v2) or risk getting some very strange results**)
- Predicting maximal growth rates when maximal growth is very slow
- Predicting the maximal growth rate of a psychrophile or thermophile

Partial Mode

What if you don't have a nice, complete genome ... can you still use `gRodon`? Yes, you can, though you may want to use either "partial" or "metagenome" mode.

When your genome is incomplete, you may have insufficient data to accurately estimate the codon pair bias (since there are many possible codon pairs). In this case you can set `gRodon` to "partial" mode, which excludes pair-bias from the prediction. This is probably a good choice when working with incomplete SAGs and MAGs. The expected decrease in accuracy is quite small (see `gRodon` paper). If you suspect your MAGs are contaminated, metagenome mode may be a better option.

```
predictGrowth(genes, highly_expressed, mode="partial")
#> Warning in filterSeq(genes = genes, highly_expressed = highly_expressed, :
#> There were 146 genes either with lengths not multiples of 3 or not above length
#> threshold (default 240bp), these genes have been ignored
#> $CUBHE
#> [1] 0.9148924
#>
#> $GC
#> [1] 0.3917167
#>
#> $GCdiv
#> [1] 0.1082833
#>
#> $ConsistencyHE
#> [1] 0.5062119
#>
#> $CUB
#> [1] 0.564834
#>
#> $CPB
#> [1] -0.3886505
#>
#> $FilteredSequences
#> [1] 146
#>
#> $nHE
```

```
#> [1] 45
#>
#> $dCUB
#> [1] -0.6197544
#>
#> $d
#> [1] 0.4061778
#>
#> $LowerCI
#> [1] 0.333661
#>
#> $UpperCI
#> [1] 0.497245
```

Metagenome Mode

For bulk estimation of the average community-wide maximal growth rate from metagenomes, our consistency statistic is not appropriate, since different organisms may prefer different codons (even when they have similar bias values). It also doesn't make much sense to calculate the pair-bias in this scenario. Thus the model for predicting the average maximal growth rate of a metagenomic sample excludes consistency and pair-bias. Importantly, metagenome mode is expected to be less accurate than the default mode, so only use this mode if you must (i.e., you have a metagenomic sample).

EVEN MORE IMPORTANTLY - if you have a metagenome (mixed community sample) you MUST use metagenome mode. Otherwise, gRodon's consistency statistic will think you have a single organism that has a bunch of highly expressed genes optimized in different ways, and gRodon will severely underestimate your maximal growth rate. If you are getting really long minimal doubling times check to make sure you are using metagenome mode.

Metagenome mode is currently being updated. The metagenome_v1 mode is the one described in the original paper, but suffers from GC-bias (unlike gRodon's full mode). We have implemented metagenome_v2 mode which corrects for this, with a preprint coming soon describing a number of benchmarking experiments.

Let's try this on one of the samples from Okie et al., as discussed in the `gRodon` manuscript:

```
path_to_metagenome <- system.file('extdata',
  'ERR2143764_fastp_prokka_scaffolds.ffn.gz',
  package = 'gRodon')
genes <- readDNAStringSet(path_to_metagenome)
highly_expressed <- grepl("ribosomal protein",names(genes),ignore.case = T)
predictGrowth(genes, highly_expressed, mode = "metagenome_v1")
#> Warning in predictGrowth(genes, highly_expressed, mode = "metagenome_v1"):
#> Provide depth_of_coverage for your ORFs for a more realistic average community
#> growth rate
#> Warning in filterSeq(genes = genes, highly_expressed = highly_expressed, :
#> There were 8290 genes either with lengths not multiples of 3 or not above
#> length threshold (default 240bp), these genes have been ignored
#> $CUBHE
#> [1] 0.7435671
#>
#> $GC
#> [1] 0.6058782
#>
#> $GCdiv
#> [1] 0.1058782
```

```
#>
#> $ConsistencyHE
#> [1] 0.6749404
#>
#> $CUB
#> [1] 0.7182513
#>
#> $CPB
#> [1] -0.2902042
#>
#> $FilteredSequences
#> [1] 8290
#>
#> $nHE
#> [1] 106
#>
#> $dCUB
#> [1] -0.03524634
#>
#> $d
#> [1] 1.716055
#>
#> $LowerCI
#> [1] 1.499674
#>
#> $UpperCI
#> [1] 1.968805
```

You may also want to incorporate a correction for the relative abundance of different organisms in this calculation (e.g., weighted metagenome mode) by specifying the `depth_of_coverage` option. This should lead to more realistic estimates of the average community-wide maximal growth rate. To use this correction provide `depth_of_coverage` as a vector of mean depths of coverage for each gene. It does not matter if these coverages are normalized since we only care about relative coverage.

First let's take a look at our read depths:

```
path_to_coverage <- system.file('extdata',
  'ERR2143764_fastp_map2ffn_counts.tsv',
  package = 'gRodon')
read_depths <- read.delim(path_to_coverage,
                          stringsAsFactors = FALSE)
depths <- read_depths$meandepth
#Make sure in the correct order
names(depths) <- read_depths$X.rname
depth_of_coverage <- depths[gsub(" .*", "", names(genes))]
head(depth_of_coverage)
#> PNFGHFNJ_00001 PNFGHFNJ_00002 PNFGHFNJ_00003 PNFGHFNJ_00004 PNFGHFNJ_00005
#>        19.3492        35.1459        32.5432        28.4502        27.3811
#> PNFGHFNJ_00006
#>        29.4147
```

And now let's run weighted metagenome mode:

```
predictGrowth(genes,
              highly_expressed,
```

```
              mode = "metagenome_v1",
              depth_of_coverage = depth_of_coverage)
#> Warning in filterSeq(genes = genes, highly_expressed = highly_expressed, :
#> There were 8290 genes either with lengths not multiples of 3 or not above
#> length threshold (default 240bp), these genes have been ignored
#> $CUBHE
#> [1] 0.6999671
#>
#> $GC
#> [1] 0.5800937
#>
#> $GCdiv
#> [1] 0.08009369
#>
#> $ConsistencyHE
#> [1] 0.636041
#>
#> $CUB
#> [1] 0.6780066
#>
#> $CPB
#> [1] NA
#>
#> $FilteredSequences
#> [1] 8290
#>
#> $nHE
#> [1] 106
#>
#> $dCUB
#> [1] -0.03238986
#>
#> $d
#> [1] 2.267608
#>
#> $LowerCI
#> [1] 1.998003
#>
#> $UpperCI
#> [1] 2.579535
```

Slow Growers

All codon-usage based predictors of maximal growth will underestimate minimal doubling times when these doubling times are very long. In other words, these predictors will often indicate that slow-growing microbes are able to grow more quickly than they actually can. See the original paper for a more in-depth discusison, but this is because codon-usage bias tends to plateau at a lower-bound in slow-growers.

What does this mean for you? Well, `gRodon` can reliably tell you if your microbe grows slowly, but it can't tell you how slowly. In practice, minimal doubling time predictions appear to be accurate up to 5 hours. If `gRodon` predicts a doubling time above 5 hours, you can confidently say your microbe grows slowly, but not quite how slowly (minimal doubling times >5hrs tend to be underestimates). In these cases `gRodon` will even warn you to be careful:

```
path_to_genome <- system.file('extdata',
  'GCF_003253775.1_ASM325377v1_cds_from_genomic.fna.gz',
  package = 'gRodon')
genes <- readDNAStringSet(path_to_genome)
highly_expressed <- grepl("ribosomal protein",names(genes),ignore.case = T)
predictGrowth(genes, highly_expressed)
#> Warning in filterSeq(genes = genes, highly_expressed = highly_expressed, :
#> There were 780 genes either with lengths not multiples of 3 or not above length
#> threshold (default 240bp), these genes have been ignored
#> Warning in predictGrowth(genes, highly_expressed): Estimated doubling time >5
#> hours. CUB signal saturates at approx. 5 hrs... gRodon may underestimate
#> doubling times above this range. Consider simply reporting as '>5hrs'. (In
#> other words, this microbe definitely grows slowly, but we can't tell you quite
#> how slowly).
#> $CUBHE
#> [1] 0.5980514
#>
#> $GC
#> [1] 0.597257
#>
#> $GCdiv
#> [1] 0.09725697
#>
#> $ConsistencyHE
#> [1] 0.5630458
#>
#> $CUB
#> [1] 0.5498015
#>
#> $CPB
#> [1] -0.3347875
#>
#> $FilteredSequences
#> [1] 780
#>
#> $nHE
#> [1] 46
#>
#> $dCUB
#> [1] -0.08775866
#>
#> $d
#> [1] 15.63557
#>
#> $LowerCI
#> [1] 11.5035
#>
#> $UpperCI
#> [1] 21.5499
```

The example above is for a *Mycobacterium leprae* genome. This organism has a doubling time around 240 hours. So we are off by an order of magnitude, but still well within the "slow grower" range.

Psychrophiles and Thermophiles

Vieira-Silva et al. showed that codon-usage based maximal growth estimators are sensitive to growth temperature. Thermophiles and psychrophiles showed consistently higher/lower growth rates respectively. We included a temperature correstion factor in `gRodon` to account for this. We caution that this correction is based on only 31 extremophile species, and that `gRodon` has primarily been validated on mesophilic organisms.

```
predictGrowth(genes, highly_expressed, temperature = 33)
#> Warning in filterSeq(genes = genes, highly_expressed = highly_expressed, :
#> There were 780 genes either with lengths not multiples of 3 or not above length
#> threshold (default 240bp), these genes have been ignored
#> Warning in predictGrowth(genes, highly_expressed, temperature = 33): Estimated
#> doubling time >5 hours. CUB signal saturates at approx. 5 hrs... gRodon may
#> underestimate doubling times above this range. Consider simply reporting as
#> '>5hrs'. (In other words, this microbe definitely grows slowly, but we can't
#> tell you quite how slowly).
#> $CUBHE
#> [1] 0.5980514
#>
#> $GC
#> [1] 0.597257
#>
#> $GCdiv
#> [1] 0.09725697
#>
#> $ConsistencyHE
#> [1] 0.5630458
#>
#> $CUB
#> [1] 0.5498015
#>
#> $CPB
#> [1] -0.3347875
#>
#> $FilteredSequences
#> [1] 780
#>
#> $nHE
#> [1] 46
#>
#> $dCUB
#> [1] -0.08775866
#>
#> $OGT
#> [1] 33
#>
#> $d
#> [1] 16.10145
#>
#> $LowerCI
#> [1] 12.36286
#>
#> $UpperCI
#> [1] 21.18681
```

If working with thermophiles or psychrophiles you might consider using gRodon's alternative training set (below) which includes more extremophiles than the default training set.

Alternative Training Sets

From Madin et al. We also include a version of gRodon trained on an alternative set of minimal doubling time estimates from a trait database compiled by Madin et al.. This is a larger training set than the set of minimal doubling times fron Vieira-Silva et al., though both training sets give very similar results. See the gRodon paper for an in-depth discussion.

```
path_to_genome <- system.file('extdata',
  'GCF_000349925.2_ASM34992v2_cds_from_genomic.fna.gz',
  package = 'gRodon')
genes <- readDNAStringSet(path_to_genome)
highly_expressed <- grepl("ribosomal protein",names(genes),ignore.case = T)
predictGrowth(genes, highly_expressed, training_set = "madin")
#> Warning in filterSeq(genes = genes, highly_expressed = highly_expressed, :
#> There were 146 genes either with lengths not multiples of 3 or not above length
#> threshold (default 240bp), these genes have been ignored
#> $CUBHE
#> [1] 0.9148924
#>
#> $GC
#> [1] 0.3917167
#>
#> $GCdiv
#> [1] 0.1082833
#>
#> $ConsistencyHE
#> [1] 0.5062119
#>
#> $CUB
#> [1] 0.564834
#>
#> $CPB
#> [1] -0.3886505
#>
#> $FilteredSequences
#> [1] 146
#>
#> $nHE
#> [1] 45
#>
#> $dCUB
#> [1] -0.6197544
#>
#> $d
#> [1] 0.364274
#>
#> $LowerCI
#> [1] 0.2976978
#>
#> $UpperCI
#> [1] 0.4483901
```

Including AOA and NOB We have added several AOA and NOB to the Madin training set (some measurements from enrichment cultures) to augment gRodon's suitability for predicting growth rates in these groups. TO use this augmented training set use `training_set="AOA_NOB`. In particular, data for the following species were added:

- *Nitrososphaera viennensis*
- *Candidatus Nitrosocosmicus hydrocola*
- *Nitrosopumilus maritimus SCM1*
- *Nitrolancea hollandica*
- *Candidatus Nitrotoga arctica*
- *Nitrococcus mobilis*
- *Nitrospina gracilis*
- *Nitrospinae watsonii*

```
path_to_genome <- system.file('extdata',
  'GCF_000349925.2_ASM34992v2_cds_from_genomic.fna.gz',
  package = 'gRodon')
genes <- readDNAStringSet(path_to_genome)
highly_expressed <- grepl("ribosomal protein",names(genes),ignore.case = T)
predictGrowth(genes, highly_expressed, training_set = "AOA_NOB")
#> Warning in filterSeq(genes = genes, highly_expressed = highly_expressed, :
#> There were 146 genes either with lengths not multiples of 3 or not above length
#> threshold (default 240bp), these genes have been ignored
#> $CUBHE
#> [1] 0.9148924
#>
#> $GC
#> [1] 0.3917167
#>
#> $GCdiv
#> [1] 0.1082833
#>
#> $ConsistencyHE
#> [1] 0.5062119
#>
#> $CUB
#> [1] 0.564834
#>
#> $CPB
#> [1] -0.3886505
#>
#> $FilteredSequences
#> [1] 146
#>
#> $nHE
#> [1] 45
#>
#> $dCUB
#> [1] -0.6197544
#>
#> $d
#> [1] 0.3579622
#>
#> $LowerCI
#> [1] 0.2933078
```

```
#>
#> $UpperCI
#> [1] 0.4393985
```