Jun 10, 2024

Anotação de genomas de fungos

DOI
dx.doi.org/10.17504/protocols.io.n92ld8r97v5b/v1

Thiago Mafra Batista[1]

[1]Universidade Federal do Sul da Bahia

bioinfo

Thiago Mafra Batista
Universidade Federal do Sul da Bahia

OPEN ACCESS

DOI: dx.doi.org/10.17504/protocols.io.n92ld8r97v5b/v1

Protocol Citation: Thiago Mafra Batista 2024. Anotação de genomas de fungos. **protocols.io https://dx.doi.org/10.17504/protocols.io.n92ld8r97v5b/v1**

Protocol status: Working
We use this protocol and it's working

Created: June 09, 2024

Last Modified: June 10, 2024

Protocol Integer ID: 101459

Keywords: genome annotation, functional annotation, gene prediction, interproscan, maker2

Abstract

Este tutorial guiará estudantes e pesquisadores a realizarem a anotação (estrutural e funcional) de genomas de fungos com o software Maker2. Iniciamos com a identificação e mascaramento das regiões repetitivas com o RepeatModeler e RepeatMasker, respectivamente. Treinamos os preditores SNAP e Augustus para aprimorarmos a predição gênica que é realizada em duas etapas. Então, conduzimos a anotação funcional dos genes codificadores de proteínas a partir da similaridade encontrada com sequências depositadas nos bancos de dados Swissprot e Trembl e com o Interproscan.

Materials

Softwares necessários para toda pipeline:

1. RepeatModeler v2.0.3 (https://www.repeatmasker.org/RepeatModeler/)
   Obs: difícil instalação, possui várias dependências.
2. RepeatMasker (https://www.repeatmasker.org/)
   Obs: difícil instalação, possui várias dependências. Utilizar os bancos Dfam 3.6 e RepBaseRepeatMaskerEdition-20181026.tar.gz
3. CEGMA v2 (https://github.com/KorfLab/CEGMA_v2)
4. BUSCO v5.4.7 (https://busco.ezlab.org/)
5. MAKER v3.01.03 (https://www.yandell-lab.org/software/maker.html)
   Obs: difícil instalação, possui várias dependências. Instalar o módulo de paralelização durante a instalação (mpiexec.openmpi)
6. SNAP (https://github.com/KorfLab/SNAP)
7 Augustus (https://github.com/Gaius-Augustus/Augustus)
8. Diamond (https://github.com/bbuchfink/diamond)
9. Interproscan 5.65-97.0 (https://www.ebi.ac.uk/interpro/download/InterProScan/)
10. fasta-splitter.pl (**https://kirill-kryukov.com/study/tools/fasta-splitter/**)

Identificação e mascaramento das regiões repetitivas

1 Antes de realizar a predição gênica, vamos identificar as famílias de repetições com o RepeatModeler e depois mascará-las no genoma com o RepeatMasker. No exemplo abaixo estou utilizando um arquivo fasta com nome contigs_Y6065_v1.fasta, os softwares instalados em /opt/apps e seus executáveis disponíveis na variável de ambiente.

Criar o banco indexado a partir do genoma montado

```
$ /opt/apps/RepeatModeler-2.0.3/BuildDatabase -name y6065
contigs_Y6065_v1.fasta
```

Rodar o RepeatModeler com 32 processadores

```
$ RepeatModeler -pa 32 -database y6065 -LTRStruct > output.txt
```

Famílias de repetições identificadas. Hora de mascará-las no genoma

```
$ RepeatMasker -pa 32 -e rmblast -lib ../repeatmodeler_run/y6065-
families.fa -dir . -small -gff contigs_Y6065_v1.fasta 1>log
```

Predição gênica

2 A predição gênica será realizada com o software ***Maker2***. O *maker* possui um preditor interno e também faz o uso de preditores externos como o *SNAP*, *Augustus* e GeneMark. Neste tutorial, vamos utilizar o ***SNAP*** e ***Augustus***. Ambos precisam ser treinados. Para treinarmos o SNAP utilizaremos o arquivo *gff* gerado como output do ***CEGMA***.

2.1 **CEGMA (Core Eukaryotic Genes Mapping Approach)**

O CEGMA utiliza um grupo de 458 proteinas altamente conservadas em diferentes espécies. Os softwares de alinhamento identificam as junções exon-intron nos genomas avaliados e seus resultados são uteis para treinar preditores e para avaliar a completude de genomas.

```
# Cria o diretório cegma_run se ele não existir e entra nele
mkdir -p cegma_run && cd cegma_run || exit 1

# Executa o CEGMA com 64 processadores

$ cegma -T 64 -g genoma.fasta -o Y6065 1>cegma.log 2>cegma.err
```

Agora vamos treinar o SNAP a partir do *gff* gerado pelo CEGMA. Abaixo todos os comandos necessários para isso.

```
# Cria o diretório snaphmm_run dentro do diretório cegma_run e
entra nele

$ mkdir -p snaphmm_run && cd snaphmm_run || exit 1

# Gera arquivo hmm de treinamento do SNAP

$ cegma2zff cegma.gff genoma.fasta
$ fathom genome.ann genome.dna -categorize 1000
$ fathom genome.ann genome.dna -export 1000 -plus
$ forge export.ann export.dna
$ hmm-assembler.pl genoma.fasta . > Y6065.cegmasnap.hmm
```

2.2 **Maker (passo 1)**

Vamos rodar o maker em duas etapas. Na primeira, habilitamos o SNAP com o arquivo *cegmasnap.hmm* e utilizamos um conjunto de proteínas de sete leveduras diferentes (*Clavispora lusitaniae*, *Kluyveromyces lactis*, *Lodderomyces elongisporus*, *Metchnikowia bicuspidata*, *Saccharomyces cerevisiae* S288C, *Spathaspora passalidarium* e *Schefferomyces stipitis*) em um único arquivo nomeado *seven_yeasts_proteins.faa*. Essas proteínas servirão de evidências para a predição gênica. O ideal seria utilizar também dados de transcriptoma, quando disponíveis.

```
#gerar os arquivos de configuração do maker. Serão gerados quarto
arquivos.
$ maker -CTL
```

Vamos modificar apenas o arquivo maker_opts.ctl. Abaixo estão as linhas modificadas:

```
#-----Genome (these are always required)
genome=/caminho/para_o_genoma/contigs_Y6065_v1.fasta.masked
#arquivo mascarado pelo repeatmasker

#-----Protein Homology Evidence (for best results provide a file
for at least one)
protein=/caminho/para_o_arquivo/seven_yeast_proteins.faa #arquivo
com as proteínas das sete leveduras mencionadas acima
#-----Repeat Masking (leave values blank to skip repeat masking)
model_org= #apagar a palavra all

#-----Gene Prediction
snaphmm=/caminho/para_o_arquivo/Y6065_snap.hmm #SNAP HMM file
protein2genome=1 #infer predictions from protein homology, 1 =
yes, 0 = no
trna=1 #find tRNAs with tRNAscan, 1 = yes, 0 = no
pred_stats=1 #report AED and QI statistics for all predictions as
well as models
min_protein=30 #require at least this many amino acids in
predicted proteins
alt_splice=1 #Take extra steps to try and find alternative
splicing, 1 = yes, 0 = no
keep_preds=1 #Concordance threshold to add unsupported gene
prediction (bound by 0 and 1)
```

Agora vamos rodar a primeira etapa do maker. Tempo estimado: 50 minutos.

```
$ mpiexec.openmpi -n 40 maker -base y6065_pass1 </dev/null>
maker_pass1.log 2>&1 &
```

Após concluída a primeira etapa, vamos gerar os arquivos *fasta* contendo as sequências codificadoras de proteínas (nucleotídeos e aminoácidos) e o arquivo *gff* com as coordenadas das predições.

```
$ fasta_merge -d y6065_pass1_master_datastore_index.log

$ gff3_merge -n -d y6065_pass1_master_datastore_index.log
```

2.3 Treinamento do SNAP

Vamos treinar o SNAP novamente, mas agora, a partir do arquivo *gff* gerado pelo *maker* no passo 1.

```
$ mkdir snap_training && cd snap_training

$ maker2zff -n pass1_all.gff
$ fathom genome.ann genome.dna -categorize 1000
$ fathom genome.ann genome.dna -export 1000 -plus
$ forge export.ann export.dna
$ hmm-assembler.pl genoma.fasta . > Y6065.makersnap.hmm
```

2.4 Treinamento do Augustus

O treinamento do Augustus será realizado em duas etapas. Vamos usar de exemplo neste tutorial o código *y6065* que deverá ser alterado de acordo com o projeto.

Primeira etapa do treinamento:

```
$ mkdir augustus_training && cd augustus_training

$ awk '{if ($2=="maker") print }' pass1_all.gff > maker_pass1.gff
$ gff2gbSmallDNA.pl pass1_all.gff genoma.fasta 2000 y6065.gbk
1>gff2gb.log 2>gff2gb.err
$ new_species.pl --species=y6065
$ etraining --species=y6065 y6065.gbk 1>etraining.log
2>etraining.err
$ randomSplit.pl y6065.gbk 200
$ mv y6065.gbk.test y6065.gbk.evaluation
$ augustus --species=y6065 y6065.gbk.evaluation >&
first_evaluate.out
```

Segunda etapa do treinamento:

```
$ randomSplit.pl y6065.gbk 1000
$ optimize_augustus.pl --species=y6065 --kfold=4 --cpus=8 --
rounds=5 --onlytrain=y6065.gbk.train  y6065.gbk.test >& log
$ etraining --species=y6065 y6065.gbk 1>etraining2.log
2>etraining2.err
$ augustus --species=y6065 y6065.gbk.evaluation >&
second_evaluate.out
```

Agora vamos avaliar as duas etapas de treinamento. O comando *grep* irá imprimir as informações na tela. Observe os valores de sensibilidade (*sensitivity*) de especificidade (*specificity*).

```
$ grep -A 22 Evaluation first_evaluate.out

$ grep -A 22 Evaluation second_evaluate.out
```

2.5 Maker passo 2

Vamos realizar uma nova predição modificando alterando apenas as informações para o *SNAP* e para o *Augustus* no arquivo *maker_opts.ctl*

```
#-----Gene Prediction
snaphmm=/caminho/para_o_arquivo/Y6065.makersnap.hmm #SNAP HMM file
augustus_species=y6065 #Augustus gene prediction species model
```

Rodando o maker no passo 2. Tempo estimado: 50 minutos.

```
$ mpiexec.openmpi -n 40 maker -base y6065_pass2 </dev/null>
maker_pass2.log 2>&1 &
```

Após concluída a segunda etapa, vamos novamente gerar os arquivos *fasta* contendo as sequências codificadoras de proteínas (nucleotídeos e aminoácidos) e o arquivo *gff* com as coordenadas das predições.

```
$ fasta_merge -d y6065_pass2_master_datastore_index.log

$ gff3_merge -n -d y6065_pass2_master_datastore_index.log
```

Vamos agora processar os cabeçalhos das sequências, inserindo um código único de cada genoma:

```
$ maker_map_ids --prefix Y6065_ --justify 4 --iterate 1
y6065_pass2.all.gff > map_ids

$ map_fasta_ids map_ids y6065_pass2.all.maker.proteins.fasta

$ map_fasta_ids map_ids y6065_pass2.all.maker.transcripts.fasta

$ map_gff_ids map_ids y6065_pass2.all.gff
```

Agora as sequências codificadoras de proteínas preditas no genoma estão prontas para serem anotadas funcionalmente.

Anotação funcional

3 Vamos anotar as sequências a partir da similaridade encontrada com proteínas depositadas no **Swissprot**. As sequências que não tiverem correspondência, serão alinhadas no **Trembl**. Além disso, vamos anotar também com o **Interproscan**. Por fim, as sequencias que não tiverem correspondência com o Trembl, serão alinhadas no banco Non-redundants (NR), mas não para anotá-las, e sim para verificarmos se há alguma correspondência e verificarmos quantas sequências são desconhecidas, ou seja, sem homologia com qualquer outra já depositada no Genbank.

Vamos utilizar o Diamond/BlastX para as buscas por similaridade entre sequencias nos bancos Swissprot, Trembl e NR. Com o Interproscan utilizaremos todos as análises disponíveis no software. Abaixo segue um *script* para automatizar todo o processo:

```
#!/bin/bash

# Caminhos para os arquivos de entrada e bases de dados
query="/caminho/para_maker_run/y6065_pass2.maker.output/y6065_pass
2.all.maker.transcripts.fasta"
sprot="/caminho/para_database/uniprot_sprot.fasta.dmnd"
trembl="/caminho/para_database/uniprot_trembl.fasta.dmnd"
nr="/caminho/para_database/nr.dmnd"
sprot_trembl="/caminho/para_database/uniprot_sprot_trembl.fasta"
#este arquivo foi gerado com o comando $cat uniprot_sprot.fasta
uniprot_trembl.fasta > uniprot_sprot_trembl.fasta

# Busca Diamond/Blastx contra Sprot
echo "Diamond/Blastx vs Sprot"
diamond blastx -q "$query" -p 64 -d "$sprot" -k 1 -e 1e-6 --
sensitive --query-cover 0.5 --subject-cover 0.5 -o
blastx_y6065_vs_sprot.tab -f 6 >> log.txt 2>> err.txt

# Filtrar hits encontrados e buscar sequências que não deram match
awk '{print $1}' blastx_y6065_vs_sprot.tab | uniq >
uniprot_hits.txt
seqkit grep -v -f uniprot_hits.txt "$query" > uniprot_nohits.fasta

# Busca Diamond/Blastx contra Trembl
echo "Diamond/Blastx vs Trembl"
diamond blastx -q uniprot_nohits.fasta -p 64 -d "$trembl" -k 1 -e
1e-6 --sensitive --query-cover 0.5 --subject-cover 0.5 -o
blastx_y6065_vs_trembl.tab -f 6 >> log.txt 2>> err.txt

# Combinar resultados de Sprot e Trembl, e buscar sequências que
não deram match
cat blastx_y6065_vs_sprot.tab blastx_y6065_vs_trembl.tab >
blastx_y6065_vs_sprot_trembl.tab
awk '{print $1}' blastx_y6065_vs_sprot_trembl.tab | uniq >
sprot_trembl_hits.txt
seqkit grep -v -f sprot_trembl_hits.txt "$query" >
uniprot_trembl_nohits.fasta

# Busca Diamond/Blastx contra NR
echo "Diamond/Blastx vs NR"
diamond blastx -q uniprot_trembl_nohits.fasta -p 64 -d "$nr" -k 1
-e 1e-6 --sensitive --query-cover 0.5 --subject-cover 0.5 -o
blastx_y6065_vs_nr.tab -f 6 >> log.txt 2>> err.txt
```

```
# Anotar proteínas
echo "Anotando as proteínas"
maker_functional_fasta "$sprot_trembl"
blastx_y6065_vs_sprot_trembl.tab
../y6065_pass2.all.maker.proteins.fasta >
y6065_proteins_annotated.fasta

# Anotar CDS
echo "Anotando as cds"
maker_functional_fasta "$sprot_trembl"
blastx_y6065_vs_sprot_trembl.tab
../y6065_pass2.all.maker.transcripts.fasta >
y6065_cds_annotated.fasta

# Anotar GFF
echo "Anotando o GFF"
maker_functional_gff "$sprot_trembl"
blastx_y6065_vs_sprot_trembl.tab ../y6065_pass2.all.gff >
y6065_annotated.gff

# Preparação para InterProScan
mkdir -p iprscan_run && cd iprscan_run
perl ~/bin/fasta-splitter.pl --n-parts 5
../y6065_proteins_annotated.fasta

# Rodar InterProScan
echo "Rodando o Interproscan"
for i in ./*.fasta; do
    echo "/opt/apps/interproscan-5.63-95.0/interproscan.sh -i $i -
iprlookup -goterms -pa --cpu 20 -f tsv"
done > interpro_jobs_to_split.txt

split -l 1 interpro_jobs_to_split.txt batch.sh_

for script in batch.sh_*; do
    bash "$script" >> ../log.txt 2>> ../err.txt &
done

# Atualizar o GFF com resultados do InterProScan
echo "Atualizando o GFF com o resultado do interproscan"
cat *.tsv > y6065_iprscan_output.tsv
ipr_update_gff y6065_annotated.gff y6065_iprscan_output.tsv >
y6065_final_annotation.gff

wait
```

```
echo "Fim"
```

Agora é hora de atribuirmos as anotações aos arquivos *fasta* e ao arquivo *gff.* É neste momento que utilizamos o arquivo *uniprot_sprot_trembl.fasta.*

Observação: o maker_funciontal_fasta consome muita memória RAM. Serão necessários mais de 100GB. Atentem-se a isso e acompanhem o consumo durante essa fase.

```
$ maker_functional_fasta
/caminho/para_database/uniprot_sprot_trembl.fasta
blastx_y6065_vs_sprot_trembl.tab
../y6065_pass2.all.maker.proteins.fasta >
y6065_proteins_annotated.fasta

$ maker_functional_fasta
/caminho/para_database/uniprot_sprot_trembl.fasta
blastx_y6065_vs_sprot_trembl.tab
../y6065_pass2.all.maker.transcripts.fasta >
y6065_cds_annotated.fasta

$ maker_functional_gff
/caminho/para_database/uniprot_sprot_trembl.fasta
blastx_y6065_vs_sprot_trembl.tab ../y6065_pass2.all.gff >
y6065_annotated.gff
```

E por fim, vamos adicionar a anotação obtida com o Interproscan ao arquivo *gff*

```
$ ipr_update_gff y6065_pass2.all.gff y6065_iprscan_output.tsv >
y6065_final_annotation.gff
```

Anotação funcional concluída!