Predicting Genes in Single Genomes with AUGUSTUS

Katharina J. Hoff and Mario Stanke
University of Greifswald, Institute of Mathematics and Computer Science,
Walther-Rathenau-Straße 47, 17487 Greifswald, Germany, Phone +49-(0)3834-420-4642,
Fax +49-(0)3834-420-4640, E-Mail katharina.hoff@uni-greifswald.de,
mario.stanke@uni-greifswald.de

This is the pre-peer reviewed version of the article "Predicting Genes in Single Genomes with AUGUSTUS", which has been published in final form at `https://currentprotocols.onlinelibrary.wiley.com/doi/full/10.1002/cpbi.57`.

Significance Statement

The sequencing of genomes of interest has become more and more practicable also to individual research groups, who have not specialized on genome annotation. The structural annotation of protein coding genes is a necessary step for most downstream analysis and the choice of methods significantly impacts annotation quality and completeness.

AUGUSTUS is a tool for gene structure prediction that has been used in numerous genome annotation projects and cited more than a thousand times. It is also part of many genome annotation pipelines. Reasons for the popularity of AUGUSTUS are an outstanding accuracy in its field and free availability. In this protocol we strive to give a concise overview of data preparation and execution steps, including some advanced/optional protocols.

1 Abstract

AUGUSTUS is a tool for finding protein coding genes and their exon-intron structure in genomic sequences. It does not necessarily require additional experimental input as it can be applied in so-called *ab initio* mode. However, extrinsic evidence from various sources such as transcriptome sequencing or the annotations of closely related genomes can be integrated in order to improve the accuracy and completeness of the annotation. AUGUSTUS can be applied to single genomes, or simultaneously to several aligned genomes. Here, we describe steps required for training AUGUSTUS for the annotation of individual genomes and the steps to do the actual structural annotation. Further, we describe the generation and integration of evidence from various sources of extrinsic evidence.

Keywords

gene prediction, AUGUSTUS, genome annotation, evidence integration, RNA-Seq

Introduction

AUGUSTUS (Hoff and Stanke, 2013; König et al., 2016; Stanke et al., 2008, 2006a,b, 2004) is a tool for finding protein coding genes and their exon-intron structure in genomic sequences. For application to genomes, it requires species-specific parameters for its underlying hidden Markov model (HMM) or conditional random field (CRF) before it will achieve high accuracy when applied to a novel genome. Training AUGUSTUS is a supervised procedure, i.e. some set of protein coding and non-coding sequences must be identified in genomic sequences of the target species before the training step. In this protocol, we describe four approaches to achieve this (see Figure 2 on page 60):

1. Short read RNA-Seq data (e.g. Illumina or short 454 reads) can be aligned individually to the target genome to obtain spliced read information (e.g. using STAR (Dobin et al., 2013), GSNAP (Wu and Nacu, 2010) or Tophat2 (Daehwan et al., 2013)). The gene prediction tool GeneMark-ET (Lomsadze et al., 2014) has a self-training procedure that refines its parameters

according to spliced read information; subsequently, GeneMark-ET produces a set of predicted genes that can be filtered for those that are supported by RNA-Seq data (Alternate Protocol 1).

2. Full length protein sequences of the target species or a close relative can be aligned to the genome using an aligner that produces valid gene structures, e.g. GenomeThreader (Gremme, 2013a) or Scipio (Keller et al., 2008) (Alternative Protocol 2).

3. Expressed Sequence Tags (ESTs), cDNA sequences or longer 454 reads can be aligned to the genome (e.g. using GMAP (Wu and Watanabe, 2005) and BLAT (Kent, 2002)) and subsequently be assembled into transcripts with putative open reading frames (e.g. using PASA (Haas et al., 2003), Alternate Protocol 3).

4. Previously existing parameter sets from other species can be used to predict genes in the target genome with AUGUSTUS, either without adjustment or as a basis for *iterative* training. The visualization of predictions facilitates the identification of a good previously existing parameter set. On this basis, predictions can subsequently be run incorporating extrinsic evidence with AUGUSTUS. Predictions with full support by extrinsic evidence are usually high quality and can serve as training genes for the novel species. We call this approach *iterative training* (Alternate Protocol 4).

The procedure for training AUGUSTUS for a novel species on an existing set of training gene structures is described in Basic Protocol 7.

Once species-specific parameters are available, AUGUSTUS can be applied to a novel genome (an overview of modes is given in Figure 1 on page 59). This can be done using the genome sequence, alone (*ab initio* mode), or providing "hints" from extrinsic evidence. Transcriptome data in general can provide evidence about the location of introns and exons. We describe the automatic generation of hints from the following transcriptomic data sources:

1. short RNA-Seq data (e.g. Illumina reads or shorter 454 reads) (Alternate Protocol 13),

2. ESTs, longer RNA-Seq data (e.g. longer 454 reads), cDNAs (Alternate Protocol 16),

3. long read RNA-Seq reads (e.g. IsoSeq, Alternate Protocol 14.

The alignment of proteins from the target species or a close relative can provide evidence about introns, coding sequence, and start and end positions of genes (Alternate Protocol 15).

AUGUSTUS has many options. Here, we specifically describe how to *sample* alternative transcripts in *ab initio* mode (Alternative Protocol 12). Alternative transcripts from extrinsic evidence can be predicted if such evidence is provided to AUGUSTUS in a hints file (Basic Protocol 18). It is possible to combine various sources of extrinsic evidence in a single AUGUSTUS run. However, in certain application cases, AUGUSTUS should be run several times with different evidence, leading initially to several sets of gene structures. We explain how such gene sets can be merged, nonredundantly (Suppl. Protocol 19). Finally, we describe data parallelization for speeding up AUGUSTUS in Suppl. Protocol 20.

How to read commands in this protocol

Printing long commands occasionally requires an artificial linebreak. For example:

bash input

```
This is a command stretching \
  two lines.
```

The trailing backslash (\) and the initial spaces in the succeeding line need not be typed. What you actually type can look like this:

bash input

```
This is a command stretching two lines.
```

STRATEGIC PLANNING

Two important things should be considered before beginning the annotation: the input data, and how to document the annotation procedure.

Acquiring Input Data

The results of the structural genome annotation as well as most downstream steps, like functional annotation and phylogenetic analyses, critically depend on the genome assembly. Even relatively small changes to the assembly often imply that all downstream analyses have to be repeated if only to obtain consistent results. As this is often laborious, seek to use *the best possible assembly* in the first place – or – postpone some downstream analyses until the assembly and gene set are finalized. Pirovano et al. suggest submitting the genome assembly to GenBank first, before starting to annotate it (Pirovano et al., 2015). It is possible, that a contamination of the assembly with foreign DNA is identified during submission and the removal of such contaminant regions would at least require substantial coordinate changes, if not changes to gene structures as well.

Repeat Masking

Eukaryotic genomes may be very rich in repetitive elements. Repeats can severely disturb gene prediction (e.g. predicting many copies of some transposon may lead to an extremely high number of genes). We therefore strongly recommend that you mask any genome to be used for gene prediction rigorously. RepeatMasker (usage is described in (Chen, 2004), Unit 4.10) is a good starting point. Consider that you might have to create a species-specific repeat library (e.g. with RepeatScout (Price et al., 2005)). We distinguish two implicit ways of storing the repeat locations for a genome sequence: *soft-masked* and *hard-masked*.

In a *soft-masked* genomic FASTA file, all parts of the genome that are likely to be repeats are written with lower case letters, while all other sequence parts are written in upper case letters. For predicting genes with AUGUSTUS, it is ideal to use a soft-masked version of the genome. Most modern transcript aligners can also handle softmasking, and it is essential to use a repeat-masked genome for transcript alignment.

File contents example: genome.soft-masked.fa

```
>contig1
AGACCTGAgagggcattggaatctTGGCCTGGAGTCCACTCAAAGGGTACATAAAC
```

In a *hard-masked* genomic FASTA file, all parts of the genome that are considered repeats are stored as unknown nucleotides with the character N. The advantage of hardmasking is that even software that is unable to handle soft repeat masking correctly is not disturbed by the repeats.

File contents example: genome.hard-masked.fa

```
>contig1
AGACCTGANNNNNNNNNNNNNNNNTGGCCTGGAGTCCACTCAAAGGGTACATAAAC
```

From the AUGUSTUS point of view, softmasking should be preferred since AUGUSTUS will still be able to assess the sequence content in repetitive regions and may compensate overmasking during gene prediction.

(Re-)Naming Sequences

File formats for handling alignment and gene prediction data need to refer to the names of sequences. In FASTA format, sequence names are stored in the *fasta header*. There is no strict definition of how long or how complex a FASTA header may be and for many purposes, it may be helpful to have a very descriptive header. In terms of alignment and gene prediction, it is more advantageous to work with short and simple sequence names. In particular, you should avoid the occurence of whitespaces and special characters in the header.

File contents example: sequence_with_complex_header.fa

```
>P01013 GENE X PROTEIN (OVALBUMIN-RELATED) | SOME MORE INFORMATION
QIKDLLVSSSTDLDTTLVLVNAIYFKGMWKTAFNAEDTREMPFHVTKQESKPVQMMCMNNSFNVATLPAE
```

File contents example: sequence_with_simple_header.fa

```
>seq1
QIKDLLVSSSTDLDTTLVLVNAIYFKGMWKTAFNAEDTREMPFHVTKQESKPVQMMCMNNSFNVATLPAE
```

Inspect all your FASTA files before using them for gene prediction for the nature of their FASTA headers. If headers are complex, change them. You can use e.g. the script `http://bioinf.uni-greifswald.de/bioinf/downloads/simplifyFastaHeaders.pl` to produce a new file with simple headers and a mapping table with old and new headers.

bash input

```
simplifyFastaHeaders.pl in.fa prefix out.fa header.map
```

Documenting the Annotation Procedure

Firstly, expect that any critical command line or processing step needs to be repeated at a (possibly much) later point in time for a number of reasons. New data could be added later, e.g. when additional RNA-Seq libraries or protein sequences of another close relative become available or when an error was made or a suboptimal option was chosen and it becomes apparent only later. Or a new program or program version is used, e.g. another protein spliced alignment program. Also, plan for the possibility that somebody else may need to repeat the work – or – perform similar work on another data set. Comment what you do for yourself and for others.

Anticipating such repetitions, one should document all steps in a way such that the annotator requires not much "manual" work. Then the time for the update is mostly dominated by the time to wait for the computer(s) to finish. One way to do this is to copy any executed command into a text file (actually, a shell script) and to comment it using `#`. For example, such a documentation file could look like this

File contents example: `metaparopt.sh`

```
# Tue Apr 17, 2018
mkdir try1; cd try1
BONAFIDE_DIR1=/path/to/training/data/version1
ln -s $BONAFIDE_DIR1/buggenes.1234.gb train.gb
ln -s $BONAFIDE_DIR1/buggenes.345.gb test.gb
# optimize meta parameters
optimize_augustus.pl --species=bug --cpus=4 --kfold=8 train.gb > optimize.out
# final training with optimized meta parameters
etraining --species=bug train.gb
# make a test prediction to assess accuracy
augustus --species=bug test.gb > test.opt.out
```

and a repeated analysis could be performed after minimal changes to above documentation file (e.g. changing try1 to try2). In fact, if above commands are stored in a text file `metaparopt.sh` it can be invoked as a shell script with the commands

bash input

```
chmod +x metaparopt.sh
metaparopt.sh
# or
bash metaparopt.sh
```

For more tips on how to organize one's data and programs we refer the reader to Noble (2009).

Necessary Resources

Hardware

The protocols below are in principle executable on a modern desktop computer with 8 GB RAM (exception: RNA-Seq alignment with e.g. the tool STAR requires at least 10 x GenomeSize bytes RAM) and little more hard drive space than required to hold the input data (e.g. genome and

RNA-Seq data). In particular for genome sizes in the multi-gigabase range, we recommend a workstation with at least 8 or more compute cores or a compute cluster. The overall running times are dominated by the time to run alignments and the time to run AUGUSTUS. For example, aligning the example Illumina data from Alternate Protocol 1 to the given genomic sequence with STAR takes ∼6 minutes on an Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz with 8 cores; if you choose a slower aligner, runtime will be higher. Expect single-species gene predictions with AUGUSTUS to take in the order of 3 days per Gb of genome if performed on a single CPU core, if the hints (evidence) are already prepared.

Software

All commands and software calls in this book chaper have been tested with the Linux `bash` shell. They may not be compatible with other operating systems.

Described protocols require the gene prediction software AUGUSTUS. Obtain the most recent release of AUGUSTUS from the developers' website `http://bioinf.uni-greifswald.de/augustus` by typing:

bash input

```
wget http://bioinf.uni-greifswald.de/augustus/binaries/augustus.current.tar.gz
```

Extract `augustus.current.tar` with:

bash input

```
tar -xvf augustus.current.tar.gz
```

Continue with installation instructions provided in file `augustus/README.TXT`.

Please note that "AUGUSTUS" consists of several binaries in `augustus/bin/` (executable tool files, e.g. `augustus`, `etraining`, `joingenes`, `bam2hints`) and a number of perl scripts in `augustus/scripts/`. Sources of the main components of AUGUSTUS are located in `augustus/src/`, sources of auxillary tools (such as `joingenes`) are located in `augustus/auxprogs/`. Depending on your system configuration, you may have to pay some attention to compilation of the tools in `augustus/auxprogs/`. The auxiliary programs have individual compilation instructions extending `augustus/README.TXT`. Not all auxiliary programs may be needed for a given application. Third party software requirements are explicitely listed above each protocol.

Training AUGUSTUS

AUGUSTUS is a gene prediction tool that utilizes either a generalized hidden-Markov model (generalized HMM) or semi-Markov conditional random field (CRF) (Stanke and Borodovsky, 2018) to identify protein-coding genes and their structure in a given genome sequence. The HMM is the default model and used in most of this chapter. In Alternate Protocol 9 we describe how to make use of the CRF model. Both models are probabilistic and the models have very roughly 10 000 parameters. The parameters are estimated (in a so-called supervised fashion) from genome sequences

with bonafide reference gene structures (the *training set*). To achieve accurate gene prediction, the parameters should match the target species (see next paragraph). Each parameter set is stored in a subfolder of the directory `augustus/config/species/`. The AUGUSTUS release already contains many pre-trained parameter sets. We highly recommend using a pre-trained parameter set if one is available for your target species, or a closely related one.

It is not necessary to train AUGUSTUS specifically and individually for each target species. A *cross-species* application may work fine when parameters were trained on one species, say species A, and are applied to predict genes in the genome of another species, say, species B. As a rule of thumb, retraining for B is not necessary if A and B are so closely related that their genomes can be aligned with each other at least in most coding regions (e.g. the average protein sequence alignment has $> 70\%$ identities, as in human and mouse, for example). Table 1 on page 57 shows an example of the loss of accuracy in cross-species training.

AUGUSTUS requires supervised training, i.e. it needs a set of reliable training gene structures including a flanking, non-coding region at both ends. The file format used for training AUGUSTUS is a strict GenBank flatfile format:

File contents example: `train.gb`

```
LOCUS       2R_5065592-5067680   2089 bp  DNA
FEATURES             Location/Qualifiers
     source          1..2089
     mRNA            complement(join(838..872,937..1252))
                     /gene="1110_t"
     CDS             complement(join(838..872,937..1252))
                     /gene="1110_t"
BASE COUNT     660 a   396 c  350 g   683 t
ORIGIN
        1 tcctagtcat gtgaatacca aagcttttcc catctgttgt tttggtcttt gcgtcaaaca
(...)
     2041 ttttaattaa aaaattaact taataataat tctcattatt ttggagagg
//
```

The AUGUSTUS package contains the script `gff2gbSmallDNA.pl` for converting GTF or GFF3 format into GenBank format. If a GenBank flatfile was constructed with this script, the `LOCUS` line describes the genomic location from which the training gene structure was excised (here sequence `2R`, genomic start position `5065592`, end position `5067680`). Subsequently, the sequence length (here `2089 bp`) and sequence type (for training AUGUSTUS always `DNA`) are displayed. The `source` sequence goes from position `1` to `2089` (which is the sequence length); an mRNA with two exons is annotated on the reverse strand. In the given example, the exons are identical to the CDS features, below, i.e. no untranslated regions are annotated in the mRNA feature. The `BASE COUNT` lists the sequence nucleotide composition. After the `ORIGIN` line, the DNA sequence begins. It ends before the `//` tag. The programs for training AUGUSTUS are adapted to this particular GenBank flatfile format and GenBank files that deviate from above format may cause problems.

An absolute minimum for acceptable performance are 200 gene structures for training. The more genes the training set comprises the better, however the incremental benefit of additional genes is small at 1000 genes already. As a rule of thumb, if you have more than 1000 genes, then rather prefer quality (fewer false annotations) over quantity. It is also important that the number of multiexon genes is large. These are needed to train the intron recognizer. The gene structures of these genes should be as accurate as possible. However, it is not necessary that they are 100%

correct, neither does the annotation have to be necessarily complete. It is more important that the start codon is correctly annotated than that the stop codon is correctly annotated.

For quality control, you should always keep a certain number of training gene structures that you will not use for training AUGUSTUS. We call this set of genes the test set. After each training step, you can apply AUGUSTUS to this test set with the new parameters and measure prediction accuracy by comparing prediction and annotation. We recommend to use at least 200 genes for testing, if a sufficient number of other genes remains for training.

For training AUGUSTUS, first execute `etraining` on the training gene set (see Basic Protocol 7). Subsequently, execute `optimize_augustus.pl` (see Suppl. Protocol 8). Optionally, try whether CRF training with `etraining` improves prediction accuracy (see Alternate Protocol 9).

Alternate Protocol 1—Generating Training Gene Structures from Short Read RNA-Seq Data

The following protocol is suitable for generating training gene structures from a genome file and short read RNA-Seq data, e.g. Illumina paired-end data or short 454 reads. The RNA-Seq data should be from the target species (cross species alignments can only be expected to work satisfactorily if the two species are highly similar, e.g. with a few percent differences in the genome).

Please be aware that the BRAKER pipeline (`http://bioinf.uni-greifswald.de/augustus/binaries/BRAKER2.tar.gz`) automates training GeneMark-ET and AUGUSTUS with short read RNA-Seq data. We will not describe the usage of BRAKER in this protocol, but you might want to consider running BRAKER instead of performing the steps listed here manually.

Required Software:

- RNA-Seq aligner, e.g. STAR (`https://github.com/alexdobin/STAR`)[1]
- Samtools (`https://sourceforge.net/projects/samtools/files/`)
- GeneMark-ET (`http://opal.biology.gatech.edu/GeneMark/`)
- BRAKER (`http://bioinf.uni-greifswald.de/augustus/binaries/BRAKER2.tar.gz`, you do not need to configure the full BRAKER installation; the scripts required in this protocol are located in folder `braker/scripts/`)

Files:

- Genome file in FASTA format; for demonstration purposes, you can use the file `BRAKER2examples/genome.fa` containing chromosome 2R of *D. melanogaster*:

  bash input

  ```
  wget http://bioinf.uni-greifswald.de/augustus/binaries/BRAKER2examples.tar.gz
  tar -zxf BRAKER2examples.tar.gz
  ln -s BRAKER2examples/genome.fa genome.fa
  ```

- File with RNA-Seq reads in FASTQ format, for demonstration purposes you can use the file `rgasp_fly_2R.fastq` which contains RNA-Seq reads of *D. melanogaster* from the RGASP

[1] Other RNA-Seq aligners, e.g. GSNAP or Tophat2 work similarly well for this protocol

assessment (Steijger et al., 2013) that map to the above mentioned genome file (when mapped with tophat2):

bash input

```
wget http://bioinf.uni-greifswald.de/augustus/binaries/tutorial/data/\
  rgasp_fly_2R.fastq.gz
ln -s rgasp_fly_2R.fastq.gz rnaseq.fastq.gz
```

Protocol:

1. Use a repeat-masked genome when aligning RNA-Seq reads. For aligning RNA-Seq reads to a genome file, build a STAR index:

   bash input

   ```
   mkdir star_genome
   STAR --runMode genomeGenerate --genomeDir star_genome --genomeFastaFiles genome.fa
   ```

   This command produces an index for running the alignment with STAR in the directory `star_genome`. Use the option `--runThreadN INT` to set the number of threads available on your system.

2. Align RNA-Seq reads (FASTQ format, gzipped) to genome file:

   bash input

   ```
   STAR --genomeDir star_genome --readFilesIn rnaseq.fastq.gz \
     --readFilesCommand zcat
   ```

   Option `--runThreadN INT` will speed up the alignment process. The command produces an alignment output file `Aligned.out.sam`.

3. Convert the sam file to bam format

   bash input

   ```
   samtools view -Sb  Aligned.out.sam  >  Aligned.out.bam
   ```

4. **Optional: Sorting bam file** (this may be a strict requirement if you use a different aligner).

   bash input

   ```
   samtools sort -n Aligned.out.bam > Aligned.out.s.bam
   samtools sort Aligned.out.s.bam > Aligned.out.ss.bam
   ```

   If you skip this step to minimize runtime (e.g. because you are working with STAR, which produces a sorted output), please create the following softlink instead:

   bash input

   ```
   ln -s Aligned.out.bam Aligned.out.ss.bam
   ```

5. **Optional: Filtering bam file**

   RNA-Seq alignment data is potentially noisy, i.e. it may contain many alignments that are rather coincidental. To eliminate some of that noise, you can apply a filter with the program `filterBam` from the AUGUSTUS package to the alignment (bam) file before converting it to hints.

   (a) Begin with the file `Aligned.out.ss.bam` from step 4. Then, you will filter the aligned reads for those that map to one position in the genome only (`--uniq`).

   bash input

   ```
   filterBam --uniq --in Aligned.out.ss.bam --out Aligned.out.ssf.bam
   ```

   If you are using paired end RNA-Seq data, you may filter for those reads where both in a pair map to the genome with options `--paired --pairwiseAlignment`.

   (b) If you are working with paired end data, a sorting step is required (the command given here will overwrite the original `Aligned.out.ss.bam` file from step 4):

   bash input

   ```
   samtools sort Aligned.out.ssf.bam -o Aligned.out.ss.bam
   ```

   If you are working with unpaired data, skip the sorting step, remove the original file `Aligned.out.ss.bam`, and create a softlink from the `Aligned.out.ssf.bam` to `Aligned.out.ss.bam`:

   bash input

   ```
   rm Aligned.out.ss.bam
   ln -s Aligned.out.ssf.bam Aligned.out.ss.bam
   ```

6. Extract intron information from bam file:

   bash input

   ```
   bam2hints --intronsonly --in=Aligned.out.ss.bam --out=introns.gff
   ```

   The following lines show an example of the file `introns.gff`, which we also refer to as a *hints* file. This file will be used in training (this section) and for prediction later on (Section: Prediction Using Extrinsic Evidence).

   File contents example: `introns.gff`

   ```
   2R      b2h     intron  336478  343473  0       .       .       mult=3;pri=4;src=E
   2R      b2h     intron  336480  343473  0       .       .       mult=11;pri=4;src=E
   2R      b2h     intron  336482  343473  0       .       .       mult=2;pri=4;src=E
   2R      b2h     intron  336658  427382  0       .       .       pri=4;src=E
   ```

7. Most RNA-Seq data is by nature unstranded, i.e. it is unclear from which strand an aligned read stems. We try to guess the correct strand by using genomic splice site information (and

discard all reads that do not have appropriate splice site information):

bash input

```
filterIntronsFindStrand.pl genome.fa introns.gff --score  > introns.f.gff
```

This produces a file `introns.f.gff` that contains strand information and the information of how many reads map to a particular splice site.

8. Run unsupervised GeneMark-ET training and prediction to generate template genes for training AUGUSTUS:

bash input

```
gmes_petap.pl --verbose --sequence=genome.fa --ET=introns.f.gff
```

Use option `--cores=N` for parallelization. Use option `--soft_mask 1000` if the genome is soft-masked. This command may be time consuming, it takes about 6 minutes on a *Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz* with 8 cores when using the example data (3.1 GB Illumina reads, 25 MB genome file). It produces an output file `genemark.gtf` that contains *ab initio* gene predictions by GeneMark-ET.

9. Next we will filter the GeneMark-ET predictions for those that have support in all introns by RNA-Seq data (single exon genes will be selected randomly in an appropriate proportion):

bash input

```
filterGenemark.pl --genemark=genemark.gtf --introns=introns.f.gff
```

This produces several files, among them `genemark.f.good.gtf`, which contains those GeneMark genes that have support by RNA-Seq alignments (and some single exon genes).

In order to make this protocol compatible with other protocols in this protocol, we link this file to `bonafide.gtf`:

bash input

```
ln -s genemark.f.good.gtf bonafide.gtf
```

10. AUGUSTUS requires not only information about the coding sequence, but also about the non-coding sequence. We provide information about non-coding sequence in the flanking region of genes in the GenBank flatfile. The flanking region should not be chosen too short. On the other hand, setting it to maximum length may increase the runtime of training a lot. As a rule of thumb, you may use half of the mRNA size. Compute this value e.g. with `computeFlankingRegion.pl`:

bash input

```
computeFlankingRegion.pl bonafide.gtf
```

The output will look similar to this:

bash output

```
The flanking_DNA value is: 1094 (the Minimum of 10 000 and 1094)
```

We are looking for the number 1094, which will be used in the next step.

11. Convert all GeneMark gene structures to GenBank flatfile format. Whenever two neighboring genes are separated by fewer than 2*flanking_DNA bases, the end of the flanking DNA of both is set to the midpoint between the two genes. Therefore, we do not use the filtered GeneMark gene structures here, because by using all of them we exclude potentially coding regions from the flanking region of AUGUSTUS training genes, which will increase specificity of AUGUSTUS:

bash input

```
gff2gbSmallDNA.pl genemark.gtf genome.fa 1094 tmp.gb
```

12. Filter those genes that are in the "good" file `bonafide.gtf` from `tmp.gb` and store them in a file `bonafide.gb`:

bash input

```
filterGenesIn_mRNAname.pl bonafide.gtf tmp.gb > bonafide.gb
```

The file `bonafide.gb` can serve as an input to the protocol for eliminating redundant training genes in Suppl. Protocol 6.

Alternate Protocol 2—Generating Training Gene Structures from Proteins

The protocol described here is suitable for generating training gene structures from a genome file and a protein sequence file. Proteins may originate from the same species as the genome or from a very closely related species (rule of thumb: at least 90% identity of aligned orthologs). You need to choose an aligner that will not only accurately align the sequences, but also produce valid full-length gene structures with correct splice sites (e.g. GenomeThreader or Scipio). Here, we use GenomeThreader because it is faster.

Required software:

- GenomeThreader (`http://genomethreader.org/`)
- `startAlign.pl` from BRAKER (`http://bioinf.uni-greifswald.de/bioinf/braker`)

Required files:

- Genome file in FASTA format; for demonstration purposes, you can use the file `chr1to3.fa` (containing chromosomes 1 to 3 of the *Rattus norvegicus* genome):

bash input

```
wget http://bioinf.uni-greifswald.de/augustus/binaries/example-data-webserver/\
chr1to3.fa
ln -s chr1to3.fa genome.fa
```

- File with protein sequences in FASTA format; for demonstration purposes you can use the file `rattusProteinsChr1to3.fa`:

bash input

```
wget http://bioinf.uni-greifswald.de/augustus/binaries/example-data-webserver/\
  rattusProteinsChr1to3.fa
ln -s rattusProteinsChr1to3.fa proteins.fa
```

Protocol:

1. Align protein sequences to target genome with GenomeThreader. `startAlign.pl` will call GenomeThreader with certain parameters and handle parallelization:

   bash input

   ```
   startAlign.pl --genome=genome.fa --prot=protein.fa --prg=gth
   ```

   Specify the number of cores with `--CPU=INT` (it will use at most as many cores as there are genome sequences, though).

   The command will produce an output file `align_gth/gth.concat.aln` that will later serve as basis for training gene structure generation.

2. Convert `align_gth/gth.concat.aln` to GTF format:

   bash input

   ```
   gth2gtf.pl align_gth/gth.concat.aln bonafide.gtf
   ```

3. Compute a flanking region length for training gene structures in GenBank flatfile format (compare to step 10 in Alternate Protocol 1):

   bash input

   ```
   computeFlankingRegion.pl bonafide.gtf
   ```

4. Convert genome file and GenomeThreader gtf training gene file to GenBank flatfile format for training AUGUSTUS (10000 here is the flanking region size):

   bash input

   ```
   gff2gbSmallDNA.pl bonafide.gtf genome.fa 10000 bonafide.gb
   ```

The file `bonafide.gb` can serve as an input to the protocol for postprocessing of training genes in Suppl. Protocol 6.

Alternate Protocol 3—Generating training gene structures from ESTs

The protocol described here is suitable for generating training gene structures from expressed sequence tags (ESTs) and the matching genome.

Required software:

- PASA v2.2.0 (`https://pasapipeline.github.io/#A_obt_pasa`)
- MySQL
- Perl module `DBD::mysql`
- GMAP (`http://research-pub.gene.com/gmap/`)
- BLAT (`http://hgwdev.cse.ucsc.edu/~kent/src/blatSrc35.zip`)
- FASTA general sequence alignment utility (`http://faculty.virginia.edu/wrpearson/fasta/CURRENT/`)

Required files:

- Genome file in FASTA format; for demonstration purposes you can use the file `augustus/examples/autoAug/genome.fa`
- File with ESTs in FASTA format; for demonstration purposes you can use the file `augustus/examples/autoAug/cdna.fa`

Protocol:

1. Clean your transcript sequences for PASA:

   bash input

   ```
   seqclean cdna.fa
   ```

   This command produces a number of output files, most importantly the file `cdna.fa.clean`.

2. Prepare a configuration file for running PASA. Start by copying the `pasa.alignAssembly.Template.txt` file from `$PASAHOMA/pasa_conf` to your working directory:

   bash input

   ```
   cp $PASA_HOME/pasa_conf/pasa.alignAssembly.Template.txt alignAssembly.config
   ```

3. Edit the file `alignAssembly.config` as follows (e.g. using any common text editor like `gedit` or `pluma` or `emacs`):

   File contents example: `alignAssembly.config`

   ```
   MYSQLDB=sample_mydb_pasa
   validate_alignments_in_db.dbi:--MIN_PERCENT_ALIGNED=0.8
   validate_alignments_in_db.dbi:--MIN_AVG_PER_ID=0.9
   subcluster_builder.dbi:-m=50
   ```

4. Run PASA using the following command[2]:

bash input

```
Launch_PASA_pipeline.pl -c alignAssembly.config -C -R -g genome.fa \
  -t cdna.fa.clean -T -u cdna.fa --ALIGNERS blat,gmap
```

Use the option `--CPU INT` for parallelization.

Among other files, this will produce `sample_mydb_pasa.assemblies.fasta` and `sample_mydb_pasa.pasa_assemblies.gff3` that we will use in the next step.

5. Call open reading frames (ORFs) in PASA assemblies as follows:

bash input

```
pasa_asmbls_to_training_set.dbi --pasa_transcripts_fasta \
  sample_mydb_pasa.assemblies.fasta --pasa_transcripts_gff3 \
  sample_mydb_pasa.pasa_assemblies.gff3
```

This produces a file `sample_mydb_pasa.assemblies.fasta.transdecoder.cds` with potential training gene ORFs and a sister file with the same ORFs in GFF3 format (`sample_mydb_pasa.assemblies.fasta.transdecoder.gff3`).

6. Create a list of complete ORFs:

bash input

```
grep complete sample_mydb_pasa.assemblies.fasta.transdecoder.cds | perl -pe \
  's/>(\S+).*/$1/' > complete.orfs
```

The file `complete.orfs` will look similar to this:

File contents example: `complete.orfs`

```
asmbl_102|m.7
asmbl_113|m.123
(...)
```

7. Find the complete ORFs in the file `sample_mydb_pasa.assemblies.fasta.transdecoder.gff3`, keep only the exon and CDS entries; rename exon entries to have the same identifier as CDS entries:

bash input

```
grep -F -f complete.orfs \
  sample_mydb_pasa.assemblies.fasta.transdecoder.genome.gff3 | grep -P \
  "(\tCDS\t|\texon\t)" | perl -pe 's/cds\.//; s/\.exon\d+//;' \
  > trainingSetComplete.gff3
```

8. Sort the training gene candidates:

[2] Troubleshooting PASA error message *DBI connect(':localhost # or server.com','pasa-all',...) failed*: delete the `# or server.com` from `$PASA_HOME/pasa_conf/conf.txt` if you want to work with `localhost`.

bash input

```
mv trainingSetComplete.gff3 trainingSetComplete.temp.gff3
cat trainingSetComplete.temp.gff3 | perl -pe 's/\t\S*(asmbl_\d+).*/\t$1/' \
  | sort -n -k 4 | sort -s -k 9 | sort -s -k 1,1 > trainingSetComplete.gff3
```

In order to make this protocol compatible with other protocols in this protocol, we link this file to `bonafide.gtf`:

bash input

```
ln -s trainingSetComplete.gff3 bonafide.gtf
```

9. Follow steps 3 and 4 of Alternative Protocol 2.

The resulting file `bonafide.gb` can serve as an input to the protocol for postprocessing of training genes in Suppl. Protocol 6.

Alternate Protocol 4—Generating Training Gene Structures from Iterative Gene Prediction

This protocol describes a method to construct a training set using AUGUSTUS itself and extrinsic evidence data. In summary, AUGUSTUS is used to predict genes with already existing species-specific parameters (from another species) using extrinsic evidence. Then the predictions are filtered to obtain training genes (see block D in Figure 2 on page 60).

Required files:

- File with genome sequence in FASTA format,
- File with extrinsic evidence hints for gene prediction in GFF format.

In principle you can use all file combinations listed in Section: Prediction Using Extrinsic Evidence for generating hints files from extrinsic evidence, e.g. the files for generating RNA-Seq hints for *D. melanogaster* in Alternate Protocol 1.

Protocol:

1. Find an appropriate existing AUGUSTUS parameter set.

   Tip: Cross-species training, in which the parameters estimated from one species are used to predict genes in another, may give acceptable results. If the two species are close, it may not be necessary to have a separate parameter set (example: human and mouse). As a rule of thumb - if large parts of the genomes are alignable then it is not necessary to retrain. Prediction on the zebrafish genome with human parameters results roughly in a loss of 8 percent points on exon level sensitivity (from 80%). If all available parameters are from very distant species only, one may have to try a few. The closest species does not necessarily give the best results.

   Run AUGUSTUS with a few existing parameter sets:

bash input

```
for s in nasonia tribolium2012 parasteatoda honeybee1
do
  augustus --species=$s chr2L.sm.fa --softmasking=on --predictionEnd=1000000 \
    > aug.$s.1-1M.gff &
done
```

Above command will run AUGUSTUS in the background with four different parameter sets on the first megabase of `chr2L` of *Drosophila melanogaster*. The species selection is just an example. For *Drosophila* and many other insects there are already parameter sets (in this case `fly`), which we chose to ignore for the purpose of this protocol. The options `--predictionEnd` and `--predictionStart` can be used to cut out a window for prediction. We do this here to save time and as we will only visually inspect a small region. The files `aug.nasonia.1-1M.gff`, `aug.tribolium2012.1-1M.gff`, `aug.parasteatoda.1-1M.gff` and `aug.honeybee1.1-1M.gff` now contain the coordinates and protein sequences of predicted genes.

File contents example: `aug.nasonia.1-1M.gff`

```
(...)
chr2L AUGUSTUS gene         6693  9276  0.51  +  . g1
chr2L AUGUSTUS transcript   6693  9276  0.51  +  . g1.t1
chr2L AUGUSTUS start_codon  6693  6695  .     +  0 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS intron       6809  7573  0.94  +  . transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS intron       8117  8192  1     +  . transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS intron       8590  8667  1     +  . transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS          6693  6808  0.57  +  0 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS          7574  8116  0.96  +  1 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS          8193  8589  1     +  1 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS          8668  9276  1     +  0 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS stop_codon   9274  9276  .     +  0 transcript_id "g1.t1"; gene_id "g1";
# protein sequence = [MMKRDRFYWDKNDRSRERSGLNFTLTKMLAIQNGGGMKSNREENPRLNKYRRVDNSY (...)
```

2. Visualize alignments and gene predictions in a browser. For the purpose of this protocol we will assume that the extrinsic evidence data is RNA-Seq alignment data in bam-format (e.g. produced by hints from RNA-Seq generation in Alternate Protocol 13). Here, we will use the UCSC genome browser, which conveniently already exists for the genome assembly that we use (dm6). There will usually be no browser for a new genome, but nevertheless the UCSC genome browser may be used to display the genome and other tracks using *Track Hubs*. Make "big" indexed files:

bash input

```
echo "chr2L   23513712" > chrom.sizes # size of chromosomes
mv Signal.Unique.str1.out.wig rnaseq.wig    # rename for convenience
mv Aligned.sortedByCoord.out.bam rnaseq.bam
wigToBigWig rnaseq.wig chrom.sizes rnaseq.bw
bamtools index -in rnaseq.bam # index required by UCSC browser
scp rnaseq.bw rnaseq.bam rnaseq.bam.bai \
  user@server:~/public_html/ # copy to publicly accessible web space
```

The data is now web-accessible through an URL, for example, the URL could be `http:`

//hgwdev.cse.ucsc.edu/~mario/tutorial2015/results/. The UCSC browser will load the parts requested through browsing only. Create a custom track file:

bash input

```
echo "browser hide all" > customtrack
echo "track name=\"STAR RNA-Seq alignments\" type=bam visibility=4 \
  bigDataUrl=http://hgwdev.cse.ucsc.edu/~mario/tutorial2015/results/rnaseq.bam" \
  >> customtrack
echo "track name=\"STAR coverage\" type=bigWig visibility=2 \
  bigDataUrl=http://hgwdev.cse.ucsc.edu/~mario/tutorial2015/results/rnaseq.bw" \
  >> customtrack

for s in nasonia tribolium2012 parasteatoda honeybee1; do
   echo "track name=\"AUGUSTUS $s abinitio\"  db=dm6 visibility=3" >> customtrack
   grep -P "AUGUSTUS\t(CDS|exon)\t" aug.$s.1-1M.gff \
     >> customtrack # use only the relevant coordinate lines
done
```

A format example of file customtrack is shown in shown below:

bash input

```
grep -B 1 -A 2 track customtrack
```

bash output

```
browser hide all
track name="STAR RNA-Seq alignments" type=bam visibility=4 \
   bigDataUrl=http://hgwdev.cse.ucsc.edu/~mario/tutorial2015/results/rnaseq.bam
track name="STAR coverage" type=bigWig visibility=2 \
   bigDataUrl=http://hgwdev.cse.ucsc.edu/~mario/tutorial2015/results/rnaseq.bw
track name="AUGUSTUS nasonia abinitio"  db=dm6 visibility=3
chr2L AUGUSTUS CDS   6693    6808    0.57 + 0 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS   7574    8116    0.96 + 1 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS   995604  995891  0.59 + 0 transcript_id "g159.t1"; gene_id "g159";
track name="AUGUSTUS zebrafish abinitio"  db=dm6 visibility=3
chr2L AUGUSTUS CDS   7574    8116    0.98 + 1 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS   8193    8589    0.85 + 1 transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS   957279  957449  1    + 0 transcript_id "g135.t1"; gene_id "g135";
track name="AUGUSTUS tomato abinitio"  db=dm6 visibility=3
chr2L AUGUSTUS exon  6486    6808    .    + . transcript_id "g1.t1"; gene_id "g1";
chr2L AUGUSTUS CDS   6591    6808    0.76 + 0 transcript_id "g1.t1"; gene_id "g1";
```

This is a format that the UCSC Genome Browser understands. The first line tells it to hide the existing tracks. The first two track lines refer to the RNA-Seq tracks and specify the URL where the data lies. The AUGUSTUS tracks are included in the file as they are small enough.

Next:

(a) open the browser for *Drosophila melanogaster* (http://genome.ucsc.edu/cgi-bin/hgTracks?db=dm6),

(b) click on "add custom tracks" or "manage custom tracks",

(c) upload the file `customtrack` and

(d) click on the link in the column "Pos" in the table of custom tracks.

For demonstration purposes, you find a previously prepared custom track at `http://genome.ucsc.edu/cgi-bin/hgTracks?db=dm6&hgt.customText=http://hgwdev.cse.ucsc.edu/~mario/tutorial2015/results/customtrack.gz`.

The UCSC Genome Browser group has an excellent help page for setting up UCSC custom tracks.

Browse the predictions and alignments and **pick a parameter set** for which the genes match the evidence best. Let us assume here that the `nasonia` parameter set is best.

3. Predict the genes again using the best parameters and available hints as described in Basic Protocol 18, step 3. This results in a file `augustus.hints.gff`. This file contains gene structures and also a summary for each transcript as to how well it matches the evidence:

File contents example: `augustus.hints.gff`

```
...
chr2L   AUGUSTUS    transcript      11215   17136   1     -     .     g2.t1
...
# Evidence for and against this transcript:
# % of transcript supported by hints (any source): 100
# CDS exons: 8/8
#      W:   8
# CDS introns: 7/7
#      E:   7
```

4. Filter for genes with 100% support. Be aware: We are likely to introduce a bias when we restrict ourselves to this set, e.g. to the highly expressed genes. Also, we are introducing a bias towards genes that fit the previous model well. We apply the following very simple filter: Select all transcripts that are fully supported by evidence. Here we are working with RNA-Seq data. We choose to use genes that are in every intron supported by an exactly matching RNA-Seq alignment gap and that have some coverage in every exon:

bash input

```
cat augustus.hints.gff | perl -ne 'if (/\ttranscript\t.*\t(\S+)/){$tx=$1;} \
  if (/transcript supported.*100/) {print "$tx\n";}' | tee supported.lst | wc -l
```

5. Compute flanking region size (see also step 10 in Alternate Protocol 1):

bash input

```
grep -P "\tAUGUSTUS\t" augustus.hints.gff > augustus.hints.gtf
computeFlankingRegion.pl augustus.hints.gtf
```

6. Create a training set containing only the supported genes:

bash input

```
gff2gbSmallDNA.pl --good=supported.lst augustus.hints.gff genome.fa 1094 \
  bonafide.gb
```

The file `bonafide.gb` can serve as an input to the protocol for postprocessing of training genes in Suppl. Protocol 6. For compatibility with this section create the followig softlink:

bash input

```
ln -s augustus.hints.gtf bonafide.gtf
```

Supplemental Protocol 5—Preparing UTR training examples

A species-specific parameter set for AUGUSTUS may optionally contain parameters for predicting untranslated regions (UTRs). Such parameters are of particular importance for the integration of transcriptome data into gene prediction. Coverage information of RNA-Seq data will not only cover coding regions, but also UTRs. The absence of UTR parameters during gene prediction with such data may lead to false positive coding sequence prediction in longer UTRs because coverage exceeds the coding sequence and stretches into UTRs.

Training genes with UTRs can in principle be generated from aligned gene expression data, such as ESTs, RNA-Seq, or Iso-Seq. The pipeline `augustus/scripts/AutoAug.pl` provides an interface for generating UTR training examples from ESTs. However, this method is currently neither suitable for unassembled RNA-Seq alignment data (since not necessarily any single read spans a complete UTR), nor for assembled RNA-Seq data (because assembly will collapse the information of the point at which most reads end into a single transcript). We intend to publish a method for automatically generating UTR training examples from RNA-Seq data soon. At this point in time, you will have to accomplish this task manually or with external software.

Here, we describe how to obtain a training file or genes with UTRs in GenBank flatfile format from gtf format. The gtf file must contain CDS features for all genes and should in addition contain 5'- and 3'-UTR features for a number of genes. Format example (contents of columns 2, 6, and 8 are irrelevant, but included to conform to the gtf format):

File contents example: `genes_with_utrs.gtf`

```
s1 manual   5'-UTR 2530693 2530700 . + 0  gene_id "g1.t1"; transcript_id "g1";
s1 manual   5'-UTR 2530735 2530772 . + 0  gene_id "g1.t1"; transcript_id"g1";
s1 AUGUSTUS CDS    2530773 2530830 . + 0  gene_id "g1.t1"; transcript_id "g1";
s1 AUGUSTUS CDS    2530893 2531019 . + 2  gene_id "g1.t1"; transcript_id "g1";
s1 AUGUSTUS CDS    2531114 2531422 . + 1  gene_id "g1.t1"; transcript_id "g1";
s1 AUGUSTUS CDS    2531483 2531588 . + 1  gene_id "g1.t1"; transcript_id "g1";
s1 manual   3'-UTR 2531592 2531937 . + 0  gene_id "g1.t1"; transcript_id "g1";
```

For obtaining good UTR parameters, you should have at least 200 training gene structures with both 3'- and 5'-UTR features.

Required files:

- A genome file in FASTA format, e.g. `genome.fa`:

bash input

```
wget http://bioinf.uni-greifswald.de/augustus/binaries/\
example-data-utr-training/genome.fa.gz
gunzip genome.fa.gz
```

- A file with gene structures including UTRs in GTF format, e.g. `genes_with_utrs.gtf`:

bash input

```
wget http://bioinf.uni-greifswald.de/augustus/binaries/\
example-data-utr-training/genes_with_utrs.gtf.gz
gunzip genes_with_utrs.gtf.gz
```

The example files listed above consist of chromosome ch2R from *Drosophila melanogaster* and corresponding annotated NCBI genes obtained from the UCSC genome browser.

Protocol:

1. In case the file `genes_with_utrs.gtf` does not exclusively contain genes that have both 5'- and 3'-UTR present, you should first identify those genes that have both features:

bash input

```
cat genes_with_utrs.gtf | perl -ne ' \
if(s/.*\t(\S+UTR)\t.*transcript_id \"(\S+)\".*/$2\t$1/){print $_;}' \
| sort -u | perl -ne '@t = split(/\t/); split; print "$t[0]\n" \
if ($g eq $t[0]); $g = $t[0];' > bothutr.lst
```

The file `bothutr.lst` contains the IDs of transcripts that have both UTRs:

File contents example: `bothutr.lst`

```
NM_001014500.3.t1
NM_001014501.3.t1
NM_001014502.2.t1
(...)
```

2. Subsequently, convert gtf and FASTA-file to GenBank flatfile format for training AUGUSTUS, keeping only those genes in the final output that are in the file `bothutr.lst`:

bash input

```
gff2gbSmallDNA.pl genes_with_utrs.gtf genome.fa 1000 --good=bothutr.lst \
bonafide.utr.gb
```

Determination of flanking region size (here 1000bp) is described in Alternate Protocol 1, step 10.

Supplemental Protocol 6—Removing Redundant Gene Structures

Training gene structures (generated from any source) may be redundant on amino acid level. Non-redundancy is important if you plan to use part of it for training and another part for assessing prediction accuracy. If genes with an almost identical amino acid sequence are annotated in two different sequences, then delete one copy. We recommend the following criterion: No two genes in the set are more than 80% identical on the amino acid level. The non-redundancy is very important to avoid overfitting during `optimize_augustus.pl`. It is of course also very important if you want to test the accuracy on a test set.

Required software:

- NCBI-Blast+ (`ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/`)

Required files:

As input, you can use any training gene set `bonafide.gtf` or `bonafide.gb` from previous sections.

Protocol:

1. **Generating a protein FASTA file matching `bonafide.gb` GenBank file.** If you obtain this file by other means, you can skip to step 2.

   (a) Depending on the way that you generated the file `bonafide.gb`, you might have fewer genes in `bonafide.gb` than in `bonafide.gtf`. At first, we will pull the transcript names of training genes from `bonafide.gb`:

   bash input

   ```
   cat bonafide.gb | perl -ne 'if(m/\/gene=\"(\S+)\"/){ \
     print "\"".$1."\"\n";}' | sort -u > traingenes.lst
   ```

   The output will contain the strings that are used as transcript names in the `bonafide.gtf` file from which `bonafide.gb` was originally generated, with quotation marks:

   File contents example: `traingenes.lst`

   ```
   "NC_000069.5_t_gene97_mRNA105"
   "NC_000069.5_t_gene98_mRNA106"
   (...)
   ```

   (b) Next, we isolate the genes in `traingenes.lst` from `bonafide.gtf`:

   bash input

   ```
   grep -f traingenes.lst -F bonafide.gtf > bonafide.f.gtf
   ```

   If all genes from `bonafide.gtf` are present in `bonafide.gb`, you can skip the two steps above and simply create a softlink from `bonafide.gtf` to `bonafide.f.gtf`:

bash input

```
ln -s bonafide.gtf bonafide.f.gtf
```

(c) Convert training gene structure gtf file and FASTA file to a FASTA file containing protein sequences. You can use the script `gtf2aa.pl` from `AUGUSTUS/scripts` for this. Note that this script uses only the standard translation table. However, this might not have too negative effects even if the target species actually uses a different genetic code table since the only operation that will be performed with the amino acid file is a BLAST process with the purpose of identifying sequence redundancy:

bash input

```
gtf2aa.pl genome.fa bonafide.f.gtf prot.aa
```

The output file `prot.aa` is in FASTA format:

File contents example: `prot.aa`

```
>NC_000069.5_t_gene97_mRNA105
CQLICEKPFLEIDDAMRSFAEKVFASEVKDEGGRHEISPFDVDEICPISLHEMQAHIFHMENLSMDGRRKRRF
(...)
```

2. Next, BLAST all training gene amino acid sequences against themselves and output only those protein sequences that are less than 80% redundant with any other sequence in the set:

bash input

```
aa2nonred.pl prot.aa prot.nr.aa
```

This is a computationally expensive step, it may take hours or even days on a single core. Adjust the number of cores available on your machine with `--cores=INT`.

The output file `prot.nr.aa` consists of protein sequences in FASTA format.

3. In file `prot.nr.aa`, we will now find a nonredundant subset of genes. Delete the leading `>` characters of their names and store them in a file `nonred.lst`:

bash input

```
grep ">" prot.nr.aa | perl -pe 's/>//' > nonred.lst
```

The format of `nonred.lst` is:

File contents example: `nonred.lst`

```
NC_000068.6_t_gene170_mRNA187
NC_000069.5_t_gene130_mRNA140
(...)
```

4. For filtering the file `bonafide.gb`, we will need a list with loci names instead of gene names. Therefore, we parse corresponding loci names and gene names from `bonafide.gb` and store

them in a file `loci.lst`:

bash input

```
cat bonafide.gb | perl -ne '
  if ( $_ =~ m/LOCUS\s+(\S+)\s/ ) {
    $txLocus = $1;
  } elsif ( $_ =~ m/\/gene=\"(\S+)\"/ ) {
    $txInGb3{$1} = $txLocus
  }
  if( eof() ) {
    foreach ( keys %txInGb3 ) {
      print "$_\t$txInGb3{$_}\n";
    }
  }' > loci.lst
```

The format of `loci.lst` is:

File contents example: `loci.lst`

```
NC_000069.5_t_gene138_mRNA148   NC_000069.5_139311597-139314090
NC_000068.6_t_gene89_mRNA98     NC_000068.6_77128981-77131243
(...)
```

The left entry is a gene name, followed by a tabulator, the right entry is the corresponding locus name.

5. `loci.lst` now contains all loci in `bonafide.gb`. Next, we find those loci that correspond to genes in `nonred.lst` and store them in `nonred.loci.lst`:

bash input

```
grep -f nonred.lst loci.lst | cut -f2 > nonred.loci.lst
```

The format of `nonred.loci.lst` is:

File contents example: `nonred.loci.lst`

```
NC_000068.6_77128981-77131243
NC_000067.5_75207158-75211273
(...)
```

6. Filter `bonafide.gb` to keep only those loci that are contained in `nonred.loci.lst`:

bash input

```
 filterGenesIn.pl nonred.loci.lst bonafide.gb > bonafide.f.gb
```

7. **Optional:** To check how many genes were filtered out, run:

bash input

```
grep -c LOCUS bonafide.gb bonafide.f.gb
```

The output could be similar to this:

bash output

```
bonafide.gb:558
bonafide.f.gb:515
```

In this example, 43 of the original training genes were discarded due to sequence redundancy on amino acid level.

8. As a preparation for the next part of this protocol (Basic Protocol 7), move `bonafide.f.gb` to `bonafide.gb` (overwriting `bonafide.gb`):

bash input

```
mv bonafide.f.gb bonafide.gb
```

Basic Protocol 7—Training AUGUSTUS for a New Species

In this section, we describe how to generate and optimize species-specific parameters for AUGUSTUS on the basis of a training gene file in GenBank flatfile format.

Required files:

- The input needs to be in compatible GenBank flatfile format (see beginning of Training AUGUSTUS). For demonstration purposes, we will here use the file `bonafide.gb` from Alternative Protocol 2 after processing according to Suppl. Protocol 6.

Protocol:

1. Create template parameter files for the new species. Let us assume your species is called *bug*:

bash input

```
new_species.pl --species=bug
```

This command creates a directory called bug under `$AUGUSTUS_CONFIG_PATH` and therein the following files:

bash input

```
ls $AUGUSTUS_CONFIG_PATH/species/bug
```

bash output

```
bug_metapars.cfg
bug_metapars.cgp.cfg
bug_metapars.utr.cfg
bug_parameters.cfg
bug_weightmatrix.txt
```

2. **Optional**: In case the organism has a non-standard translation code, AUGUSTUS can be told to use a different translation table, in particular one with a different set of stop codons. This is useful for a small number of species such as *Tetrahymena thermophilia*, in which some codons translate to a different amino acid than usual. If you train AUGUSTUS for such a species, set the variable `translation_table` in the parameter file of your species (if that would be the case for species *bug*, that would be the file `bug_parameters.cfg` of which an example is shown Suppl. Protocol 8.

   Furthermore, adjust the stop codon probabilities in the same config file. In the case of *Tetrahymena*, where `taa` and `tag` are coding for glutamine (Q), the parameter file would contain the following entries:

   File contents example: `tetrahymena_parameters.cfg`

   ```
   translation_table 6
   /Constant/amberprob 0 # Prob(stop codon = tag),
                         # if 0 tag is assumed to code for amino acid
   /Constant/ochreprob 0 # Prob(stop codon = taa),
                         # if 0 taa is assumed to code for amino acid
   /Constant/opalprob 1  # Prob(stop codon = tga),
                         # if 0 tga is assumed to code for amino acid
   ```

   Choose the translation table number according to table 2. `translation_table=1` is the default value, the standard known from text books and has standard stop codons `taa`, `tga`, `tag`. If you have a species with the standard genetic code you need not do anything.

3. The tool `etraining` creates/updates parameter files for exon, intron and intergenic region in the parameter directory of your species, e.g. `$AUGUSTUS_CONFIG_PATH/species/bug`. It will generate or update the files

   - `bug_exon_probs.pbl`
   - `bug_igenic_probs.pbl`
   - `bug_intron_probs.pbl`

   Results of our first two runs of `etraining` will not serve the purpose of actually training AUGUSTUS but (a) to adjust the parameter `stopCodonExcludedFromCDS` in the file `bug_parameters.cfg` correctly and (b) to remove erroneous gene structures that are impossible to predict for AUGUSTUS (this will keep error log files small, later).

   Some tools and databases consider the stop codon to be part of the coding sequence (CDS), some do not, resulting in a 3 bp difference in the position of the end coordinate. The parameter `stopCodonExcludedFromCDS` sets this choice, both when reading input for training and when creating output during prediction. If you do not know which standard your training set adheres to, a convenient way is to try it out by letting `etraining` report corresponding errors. Run `etraining`:

   bash input

   ```
   etraining --species=bug bonafide.gb &> bonafide.out
   ```

Count how many times the output file `bonafide.out` contains the expression `Variable stopCodonExcludedFromCDS set right` that is actually part of an error message:

bash input

```
grep -c "Variable stopCodonExcludedFromCDS set right" bonafide.out
```

bash output

```
33
```

We need to compare this number of errors to the total number of training genes in order to assess whether those 33 genes are a minority or a majority of all genes:

bash input

```
grep -c LOCUS bonafide.gb
```

bash output

```
558
```

This will produce the number of training gene structures in `bonafide.gb`, in this case 558. Compared to this number, 33 is a minority and we do not need to change the parameter `stopCodonExcludedFromCDS`. If a majority of genes reported the error "`Terminal exon doesn't end in stop codon. Variable stopCodonExcludedFromCDS set right?`", the file `bug_parameters.cfg` must be edited as follows:

File contents example: `bug_parameters.cfg`

```
(...)
stopCodonExcludedFromCDS true # default value was originally false
(...)
```

Next, you identify all those training genes that still report errors after correctly setting `stopCodonExcludedFromCDS`. These are genes that do not adhere to the standard, e.g. because they have an in-frame stop codon, or the splice site pattern is not `gt-ag` or they are syntactically incorrect (e.g. exons of negative length). Genes that produce such an error can in principle be partially used for training. However, usually it is not worthwhile to keep a small percentage of such "bad" training genes. Note, however, that this is a point where you can detect systematic errors in training set construction; for example, when `bonafide.gb` was constructed with a genome and gff file that do not match (e.g. because the assembly has been updated). Another reason for getting a suspiciously high percentage of genes with errors may be that the program that generated training gene structures does not adhere to the format standard or admits in-frame stop codons by design. In such cases, it is advisable to "manually" check a sample of simple gene structures with errors in order to find the reason.

Run `etraining` again and search those sequences that lead to errors:

bash input

```
etraining --species=bug bonafide.gb 2>&1 | grep "in sequence" | \
  perl -pe 's/.*n sequence (\S+):.*/$1/' | sort -u > bad.lst
```

Now, remove the bad genes from `bonafide.gb`:

bash input

```
filterGenes.pl bad.lst bonafide.gb > bonafide.f.gb
```

Make sure you still have a sufficient amount of training genes to proceed with training AUGUSTUS:

bash input

```
grep -c LOCUS bonafide.gb bonafide.f.gb
```

This might e.g. produce the following output:

bash output

```
bonafide.gb:558
bonafide.f.gb:352
```

If you are satisfied with the number of 352, move the file `bonafide.f.gb` to `bonafide.gb` to proceed with this demonstration:

bash input

```
mv bonafide.f.gb bonafide.gb
```

4. Create a holdout test set.

bash input

```
randomSplit.pl bonafide.gb 200
mv bonafide.gb.test test.gb
mv bonafide.gb.train train.gb
```

These commands will randomly partition the file `bonafide.gb` with trusted annotated genes into a test set `test.gb` with 200 genes and a training set `train.gb` with the remaining genes for the purpose of later accuracy evaluation. The idea is that the genes in `test.gb` are neither used for parameter estimation nor for meta parameter optimization in order to avoid a bias, e.g. through overfitting.

5. Train parameters on the basis of `train.gb` (this ensures that the holdout test set was not used to estimate parameters):

bash input

```
etraining --species=bug train.gb &> etrain.out
```

6. Adjust stop codon frequencies: the file `etrain.out` reports the frequencies of different stop codons observes in the training gene set.

bash input

```
tail -6 etrain.out | head -3
```

bash output

```
tag: 1042 (0.244)
taa: 1579 (0.37)
tga: 1647 (0.386)
```

Open the file `bug_parameters.cfg` in a text editor and adjust the stop codon frequency lines according to the output for `etraining`:

File contents example: `bug_parameters.cfg`

```
/Constant/amberprob                0.244  # Prob(stop codon = tag)
/Constant/ochreprob                0.37   # Prob(stop codon = taa)
/Constant/opalprob                 0.386  # Prob(stop codon = tga)
```

7. Run AUGUSTUS with newly trained parameters:

bash input

```
augustus --species=bug test.gb > test.out
```

This command will predict the genes on the set of test genomic sequences, compare the predictions to the reference annotation in `test.gb` and report accuracy values. The accuracy values at the end can look like this (shortened):

File contents example: `test.out`

```
*******      Evaluation of gene prediction     *******
                | sensitivity | specificity |
nucleotide level |       0.975 |        0.89 |

           | sensitivity | specificity |
exon level |       0.839 |       0.775 |

transcript | #pred | #anno |   TP |   FP |   FN | sensitivity | specificity |
gene level |   236 |   200 |  106 |  130 |   94 |        0.53 |       0.449 |
```

These numbers mean, for example, that of the 200 genes, 106 were predicted exactly (true positives: all boundaries, no missing or extra exons), 83.9% of the exons were predicted correctly (both boundaries) and 77.5% of the predicted exons were exactly as in the reference annotation. *Tip:* A common mistake can be discovered, if the gene level accuracy is 0 or very close to 0 and the exon level accuracy is reasonably high, say 60% or more. In this case, a likely explanation is that the `test.gb` and `augustus` disagree with regard to whether the stop codon is considered to be part of the CDS or not. In this case, change the setting of the variable stopCodonExcludedFromCDS and rerun above test.

Supplemental Protocol 8—Meta Parameter Optimization

This section describes an optional but recommended step: meta parameter optimization with `optimize_augustus.pl`. `etraining` estimates typically in the order of 10 000 parameters. Among these are for example the probability of the nucleotide `A` at the first position in a codon, if the four preceeding nucleotides were `GTGC`; and the probability that an internal CDS has length 57 or a probability of a specific pentamer pattern around an acceptor splice site. The number of parameters, their meaning or the estimation approach are actually controlled by a smaller number of *meta parameters* that is in the dozens. An example of such a meta parameter is the length of the patterns, i.e. for example whether to consider pentamers, hexamers or heptamers around acceptor splice sites. Another example are meta parameters that influence the smoothing of empirical distributions like length distributions.

The meta parameters are stored along other settings in the file `bug_parameters.cfg` in this example where the species is called "bug". An excerpt of such a parameters file is shown in figure 3. For a given choice of meta parameters, the above introduced program `etraining` estimates the parameters. The default meta parameters may not quite be optimal for any genome and training set. For example, this could be because the branch point region may be more or less informative or longer or shorter depending on the species; or, because the optimal value of meta parameters related to smoothing depends on the training set size.

The script `optimize_augustus.pl` seeks to improve the setting of meta parameters with an iterative heuristic that changes the meta parameters in `bug_parameters.cfg` whenever a setting reaches a better prediction accuracy.

1. Run `optimize_augustus.pl`:

   bash input

   ```
   optimize_augustus.pl --species=bug --kfold=8 train.gb > optimize.out
   ```

   Specify the number of available cores with `--cpus INT`.

   The command shown above individually varies each meta parameter specified in the file `bug_metapars.cfg` in the range and order specified in the same file. For further information on the format, the user is referred to the instructions in the `metapars.cfg` file. For each new combination of parameters, it performs a k-fold cross-validation, i.e. it randomly partitions `train.gb` in k similarly sized parts, called *buckets*. In each iteration, it takes each bucket once as a validation set to estimate accuracy after training was performed on the union of all other buckets with the current meta parameters setting. These k training and validation steps can be done in parallel (`--cpus` specifies the number of threads to use and is best set to be an integral fraction of k). The results from the cross-validation are averaged over the k predictions and weighted averaged over the sensitivity and specificy on the base, exon and transcript level to form a single target value to maximize. Optimization stops when

   - the specified number of optimization rounds (each meta parameter is optimized once per round) is reached (default: 5 rounds),
   - in any round the target could not be improved at all, or,
   - the impatient user interrupts the optimization.

Meta parameter optimization has the potential to improve the accuracy (target value) by a few percent points and the improvement can be expected to diminish with the optimization rounds. As this protocol can be time-consuming, a user may choose to interrupt the optimization. Improvements are saved to the respective `parameters.cfg` file right after they were found. In any case, the user is required to perform a final training after this optimization step (step 2).

Tip: If above step takes longer than a few CPU core days, it may be because of one of the two reasons. Firstly, the intergenic regions flanking the genes may be very large and could then be shortened. Secondly, the number of training genes may be very large (say, many thousands). In that case, the optimization can potentially be sped up a lot at no significant accuracy loss using the option `--onlytrain=onlytrain.gb` as follows:

bash input

```
randomSplit.pl train.gb 300
optimize_augustus.pl --species=bug train.gb.test --onlytrain=train.gb.train
```

The first command randomly partitions the training genes in `train.gb` such that the output file `train.gb.test` contains 300 genes and the output file `train.gb.train` contains all remaining genes, provided that `train.gb` contains more than 300 genes. The subsequent `optimize_augustus.pl` command can be much faster for large training sets as the most time-consuming prediction step is performed on a smaller number of genes, here 300.

2. Perform the final training:

bash input

```
etraining --species=bug train.gb
```

After improved meta parameters have been found in step 1 above this second step estimates the corresponding parameters on the complete training set. It is much quicker than step 1 and should not be omitted. There is a potential risk that this step 2 is forgotten because `augustus` may be used without it. However, in this case the parameters from the last training are used, which are based on only a fraction of about $k-1/k$ of the training set and quite possibly with a bad choice of the last meta parameter to be optimized. Therefore, the user is requested *not to forget* this step.

3. Measure the accuracy (again) on the test set:

bash input

```
augustus --species=bug test.gb > test.opt.out
```

See the last step in Basic Protocol 7 for an explanation.

Alternate Protocol 9—CRF-Training

By default, `etraining` estimates the parameters of a so-called hidden-Markov model (HMM). That is a probabilistic model that assigns a probability $P_\theta^{\mathrm{HMM}}(\mathbf{G},\mathbf{S})$ to all possible gene structures $\mathbf{G}$ and

all possible genome sequences $\mathbf{S}$. θ is the collection of parameters that are stored in the respective subdirectory of `config/species` for the species. When the parameters of AUGUSTUS are trained with `etraining` and the option `--CRF=on`, a so-called conditional random field (CRF) is trained. This is another probabilistic model $P_\theta^{\mathrm{CRF}}(\mathbf{G}\,|\,\mathbf{S})$ of the *conditional* probability of any gene structure $\mathbf{G}$ *given* a genome sequence $\mathbf{S}$. In contrast to the HMM, the CRF does not model the distribution of genome sequences. This is not necessary as a genome sequence is always given as input. This fact makes it easier to perform a so-called *conditional training* whose objective is – simply speaking – to find parameters θ that make the true gene structure the most likely given the input genome, rather than making a realistic model of the distribution of $\mathbf{G}$ and $\mathbf{S}$. Readers with an interest in the theoretical background are referred to the book chapter (Stanke and Borodovsky, 2018).

From a user standpoint, the following facts are important: The CRF can be significantly *more accurate* than the HMM. However, the training is *less robust*, i.e. errors in the training set can decrease the accuracy for a CRF more than for a HMM. Errors in the training set could, for example, be training sequences that contain unannotated genes or exons, e.g. in the flanking region of another gene (false negatives). For this reason, the more robust HMM training is the default and users have to explicitly request the CRF training, which also takes longer. Whether `etraining` is run with `--CRF=on` or not (equivalent to `--CRF=off`) does not influence, how AUGUSTUS is executed later. The command lines are not affected. Actually, internally even the same function is used to estimate a gene structure (via $\hat{\mathbf{G}} = \mathrm{argmax}_{\mathbf{G}}\, P(\mathbf{G}\,|\,\mathbf{S})$).

The current human parameters that are distributed with AUGUSTUS were trained with the following command:

bash input

```
etraining --species=human train.1784.gb --CRF=on --CRF_N=2 --UTR=off
```

Here, `train.1784.gb` is a training set of 1784 randomly chosen human RefSeq gene structures and sequences from GenBank. When using CRF training on a high-quality training set of *Homo sapiens* (human), the gene level sensitivity of *ab initio* predictions is 19% for CRF training versus 14.5% for HMM training. Above command line for CRF training took 7h on a single core machine. This compares to less than a minute of running time of `etraining` in the standard HMM mode. For users with a highly accurate training set we recommend trying above CRF training as well as HMM training and comparing the *ab initio* accuracies of both.

Supplemental Protocol 10—UTR Training

UTR parameters are not required for gene prediction with AUGUSTUS. When integrating expression data, the accucary of AUGUSTUS may benefit from UTR parameters, though (reduction of false positive CDS feature prediction in long UTRs).

1. Create a holdout test set:

   bash input

   ```
   randomSplit.pl bonafide.utr.gb 200
   mv bonafide.utr.gb.test test.gb
   mv bonafide.utr.gb.train train.gb
   ```

2. Measure initial accuracy without UTR parameters:

bash input

```
augustus --species=bug test.gb > test.noUtr.out
```

The file `test.noUtr.out` holds gene prediction accuracy before UTR parameter optimization in file `test.gb` with parameters for species `bug`.

3. Optimize UTR parameters:

bash input

```
optimize_augustus.pl --species=bug --rounds=3 train.gb \
  --UTR=on --metapars=$AUGUSTUS_CONFIG_PATH/species/bug/bug_metapars.utr.cfg \
  --trainOnlyUtr=1
```

Use the option `--cpus=INT` for parallelization.

Tip: If the number of training genes is very large, this step will take a long time. In that case, the optimization can potentially be sped up a lot at no significant accuracy loss using the option `--onlytrain=onlytrain.gb` as follows (compare step 1 in Suppl. Protocol 8):

bash input

```
randomSplit.pl train.gb 200
optimize_augustus.pl --species=bug --rounds=3 train.gb.test --UTR=on \
  --metapars=$AUGUSTUS_CONFIG_PATH/species/bug/bug_metapars.utr.cfg \
  --trainOnlyUtr=1 --onlytrain=train.gb.train
```

4. Test accuracy with UTR parameters:

bash input

```
augustus --species=bug test.gb --UTR=on --print_utr=on > test.utr.out
```

5. Compare accuracy with (`test.utr.out`) and without UTR parameters (`test.noUtr.out`) to check whether UTR parameters improve prediction accuracy. If you make a general decision for gene prediction with UTRs, you may edit the following lines in file `$AUGUSTUS_CONFIG_PATH/bug/bug_parameters.cfg`:

File contents example: `bug_parameters.cfg`

```
(...)
print_utr           on
UTR                 on
(...)
```

The option `print_utr` determines the way UTR exons are reported in the output (explicit as UTR, or implicit as exon), `UTR` determines whether UTRs are predicted. If set in the parameter file, you do not need to specify those options, when calling AUGUSTUS anymore.

Predicting Genes with AUGUSTUS

Basic Protocol 11—*Ab Initio* Prediction

This section describes the structural genome annotation process when no evidence is used (other than the genome, of course). Furthermore, it introduces output options that are also available in prediction modes other than *ab initio*. Protocols for genome annotation using other evidence are described in Section: Prediction Using Extrinsic Evidence below.

Required files:

- Soft-masked genome file in FASTA format; for demonstration purposes, you can use the file `augustus/examples/autoAug/genome.fa`
- Parameter files that are suitable for the target genome (see training above). We will here use `caenorhabditis` as example, for which the parameters are avaiblabe in the distribution (in `augustus/config/species/caenorhabditis/`).

Protocol

1. *Ab initio* predict the genes:

bash input

```
augustus --species=caenorhabditis genome.fa > augustus.gff
```

File contents example: `augustus.gff`

```
# start gene g7
chrI AUGUSTUS gene         59438  60095  0.18  - .  g7
chrI AUGUSTUS transcript   59438  60095  0.18  - .  g7.t1
chrI AUGUSTUS tts          59438  59438  .     - .  transcript_id "g7.t1"; gene_id "g7";
chrI AUGUSTUS 3'-UTR       59438  59458  0.78  - .  transcript_id "g7.t1"; gene_id "g7";
chrI AUGUSTUS stop_codon   59459  59461  .     - 0  transcript_id "g7.t1"; gene_id "g7";
chrI AUGUSTUS single       59459  60046  0.66  - 0  transcript_id "g7.t1"; gene_id "g7";
chrI AUGUSTUS CDS          59459  60046  0.66  - 0  transcript_id "g7.t1"; gene_id "g7";
chrI AUGUSTUS start_codon  60044  60046  .     - 0  transcript_id "g7.t1"; gene_id "g7";
chrI AUGUSTUS 5'-UTR       60047  60095  0.23  - .  transcript_id "g7.t1"; gene_id "g7";
chrI AUGUSTUS tss          60095  60095  .     - .  transcript_id "g7.t1"; gene_id "g7";
# protein sequence = [MGQETMLINTEPQKTSDSTPTASYLRQWQQDIERKVGRIEIVAEKQYRMQKKLDGLEQKLDKNA
# EKSDVNQAKLESLLEQVMQQITKNNQSSSTAEQENKRLTSRKPVRNAVTRGRNSAEQGDDGYFGEDPLLANQDRLFGDDTKWDD
# NGLPEHLIHWDGYETNEQTKEALWDTAERQRVINETVKSMEPFDGSP]
# end gene g7
###
```

This excerpt from the output describes an unspliced gene on the reverse strand. `tts` and `tss` stand for transcription termination/start site, respectively. The numbers in the sixth colum specify the probability of the feature that is specified in the line. For example, in the AUGUSTUS model, there is a 66% probability that a coding exon goes exactly from position 59459 to position 60046 on the reverse strand of `chrI`. This file is in gene feature format (GFF). This format is defined e.g. under `https://www.ensembl.org/info/website/upload/gff.html` and relatively loose, i.e. programs like AUGUSTUS have much freedom

for concrete choices like the feature names in the third column or what to put into the last column.

Alternatively, the more standardized gff3 format (`http://gmod.org/wiki/GFF3`) can be obtained, e.g. with

bash input

```
augustus --species=caenorhabditis genome.fa --gff3=on \
         --outfile=augustus.gff3 --errfile=augustus.err
```

Above command line also demonstrates a way to specify standard output and standard error files rather than using output redirection (with ">"). This can be useful when output redirection is not possible or convenient, e.g. for a job submission engine that is used to run many such commands in parallel.

File contents example: `augustus.gff3`

```
chrI AUGUSTUS gene                    59438 60095 0.18  - .  ID=g7
chrI AUGUSTUS transcript              59438 60095 0.18  - .  ID=g7.t1;Parent=g7
chrI AUGUSTUS transcription_end_site  59438 59438 .     - .  Parent=g7.t1
chrI AUGUSTUS three_prime_utr         59438 59458 0.78  - .  Parent=g7.t1
chrI AUGUSTUS stop_codon              59459 59461 .     - 0  Parent=g7.t1
chrI AUGUSTUS CDS                     59459 60046 0.66  - 0  ID=g7.t1.cds;Parent=g7.t1
chrI AUGUSTUS start_codon             60044 60046 .     - 0  Parent=g7.t1
chrI AUGUSTUS five_prime_utr          60047 60095 0.23  - .  Parent=g7.t1
chrI AUGUSTUS transcription_start_site 60095 60095 .    - .  Parent=g7.t1
```

AUGUSTUS has many useful options that are described in the `augustus/README.TXT` file.. Here, we describe some of the most frequently used options:

- You can specify which features shall be reported in the output file: `--introns=on/off`, `--start=on/off`, `--stop=on/off`, `--cds=on/off`, `--codingseq=on/off`, `--protein=on/off`.
- `--predictionStart=A, --predictionEnd=B` allows prediction in a range from A to B in the sequence(s).
- `--UTR=on/off` influences whether untranslated regions are predicted in addition to the coding sequence. This works only for those species for which the UTR model was trained.
- `--strand=both/forward/reverse`: by default, genes are reported on both strands, but you can alter this to predict only in the forward or reverse strand.
- `--genemodel=partial`, `--genemodel=intronless`, `--genemodel=complete`, `--genemodel=atleastone` or `--genemodel=exactlyone`:
  - `partial`: allow prediction of incomplete genes at the sequence boundaries (default)
  - `intronless`: only predict single-exon genes like in prokaryotes and some eukaryotes
  - `complete`: only predict complete genes
  - `atleastone`: predict at least one complete gene
  - `exactlyone`: predict exactly one complete gene

2. The protein sequences of all predicted genes can be obtained with:

bash input

```
getAnnoFasta.pl augustus.gff
```

This command creates a file `augustus.aa` in FASTA format like this:

File contents example: `augustus.aa`

```
>g7.t1
MGQETMLINTEPQKTSDSTPTASYLRQWQQDIERKVGRIEIVAEKQYRMQKKLDGLEQKLDKNAEKSDVNQAKLESLL
EQVMQQITKNNQSSSTAEQENKRLTSRKPVRNAVTRGRNSAEQGDDGYFGEDPLLANQDRLFGDDTKWDDNGLPEHLI
HWDGYETNEQTKEALWDTAERQRVINETVKSMEPFDGSP
```

If the option `--codingseq=on` was used to call AUGUSTUS to produce `augustus.gff`, the coding DNA sequence is output in addition to the amino acid sequence. In this case, `getAnnoFasta.pl` creates a second file `augustus.codingseq` with the complete coding sequences like this:

File contents example: `augustus.codingseq`

```
>chrI.g7.t1
atgggtcaggagactatgctcatcaacacggagccgcaaaaaacttcggattccacaccgacagcatcgtatctg ...
```

Obtaining the coding sequences is e.g. necessary for analyzing selection (e.g. for computing dN/dS), or for transcriptome expression quantification from RNA-Seq.

Alternate Protocol 12: Sampling Alternative Gene Structures

With the basic protocol described above, the most likely gene structure is found for each input sequence. In this alternate protocol we describe how this gene set can be extended to additionally include genes or transcript variants that are not included in the most likely gene structures but that are also fairly likely alternatives (Stanke et al., 2006a). Obtaining alternatives is not the typical objective when obtaining a best possible genome annotation for publication. However, there are several reasons why this alternative protocol can be useful:

- In the context of individual genes of high importance manual inspection of a gene structure may be justified. Sampling may then provide alternative transcripts that could be manually interpreted for plausibility, e.g. in a manual comparative analysis.
- The gene set with alternatives has a higher sensitivity (recall, number of true positives). The lower specificity (and more false positives) can be less of an issue, e.g. when an additional filter like a homology search is applied to the gene set or when false negative genes are more problematic than false positives for a given application.
- The coding sequences can serve as a more inclusive database against which spectra from tandem mass spectrometry (MSMS) can be queried (Castellana et al., 2008).

1. In order to predict genes with alternatives in *ab initio* mode, run the following command:

bash input

```
augustus --species=caenorhabditis --alternatives-from-sampling=on \
    --minexonintronprob=0.08 --minmeanexonintronprob=0.4 --maxtracks=3
    genome.fa > augustus.gff
```

Here, `minexonintronprob` and `minmeanexonintronprob` are threshold parameters between 0 and 1 to filter out transcripts with low (mean) exon and intron probabilities. `maxtracks` is an upper limit for the number of transcripts that span any given genome position with the effect that a genome browser can in principle place the predicted transcripts in such a way that `maxtracks` rows are sufficient. The command line shown above is referred to as Option 2 in Figure 4 and Table 3 on page 59, which show an example output and the transcript numbers for a few settings of these parameters.

Prediction Using Extrinsic Evidence

This section describes the structural genome annotation process when experimental evidence is used to identify (parts of) gene structures, to uncover alternative splicing or to overall improve annotation quality.

Preparing Hints

Hints are extrinsic evidence about the location and structure of genes. Each hint is local information, associated with a particular genome region and specified in a particular GFF format. When predicting genes, AUGUSTUS can incorporate these hints, which will change the likelihood of gene structure candidates. It will tend to predict gene structures that are in agreement with the hints. Here, we describe how to automatically generate hints in the correct format from RNA-Seq, IsoSeq, protein sequence and EST data. However, other sources of extrinsic evidence are possible, too, e.g. MSMS data can be used to predict reading frames correctly, or the similarity of genome sequences with those of closely related genomes can be used to improve exon-intron structures.

Grouping Hints

Each hint can be given a group name, by specifying `group=groupname;` or `grp=groupname;` in the last column for the hint in the GFF file. In the case of alignments of longer sequences like IsoSeq (Alternate Protocol 14), proteins (Alternate Protocol 15) or ESTs, this should be used to group together all the hints coming from the alignment of the same sequence to the genome. For example, if an EST with the name est_xyz aligns to the genome with one gap suggesting an intron then the hints resulting from that alignment could look like this:

File contents example: `hints.gff`

```
HS04636    blat2hints  exonpart   500  599 .  +  .  group=est_xyz; source=E
HS04636    blat2hints  intron     600  700 .  +  .  group=est_xyz; source=E
HS04636    blat2hints  exonpart   701  900 .  +  .  group=est_xyz; source=E
```

Grouping tells AUGUSTUS that hints belong together. Ideally, all hints of a group are obeyed by a predicted transcript or the whole group of hints is ignored when making the prediction. When predicting alternative splicing (Basic Protocol 18 below), such grouping can prevent the prediction of incorrect alternative splicing variants that are chimeric mixtures of several local splicing alternatives but not supported by any alignment. If for any hint in a group no valid gene structure can exist that satisfies the hint, then the whole hint group is discarded as *unsatisfiable*. This can, for example, happen if the first and last two bases of an intron hint do not satisfy the splicing consensus (GT/AG or GC/AG) or if there is no open reading frame covering the range of a hint that suggests a coding exon in a certain region.

Prioritizing Groups

Hints or hint groups can be given a priority by specifying `priority=n;` or `pri=n` in the last column in the hint GFF file. For example:

File contents example: `hints.gff`

```
HS04636        blat2hints   exonpart   500 599 .  +  .  priority=2; src=E
HS04636        blat2hints   intron     550 650 .  +  .  priority=5; src=mRNA
```

When two hints or hint groups contradict each other, the hints with the lower priority number are ignored. This is useful if, for a specific genome, several sources of hints are available, where one source should be trusted when in doubt. For example, if a genome has already a trusted but partial annotation, the exons, coding sequences and introns of this partial annotation could be given as hints with high priority. At the same time, further hints, say from protein alignments (Alternate Protocol 15), could be input as well, but with a lower priority.

Alternate Protocol 13—Generating Hints from RNA-Seq Data

RNA-Seq alignments against the genome contain two types of features that are potentially helpful for gene prediction:

1. Spliced alignments of reads give information about the possible location of introns,
2. coverage (i.e. how many reads are aligned to a particular position in the genome) gives information about the possible location of exons.

While the integration of spliced read (intron) information usually works well, the integration of coverage (exonpart) information is not trivial. The problem is that coverage may not only be high in CDS regions, but also in UTRs and in partially retained introns. If you do not have UTR parameters for your target species, we recommend that you do not use coverage information in the gene prediction step.

Required file:

- RNA-Seq to genome alignments in bam format. A suitable example file is e.g. produced in steps 4 and 5 in Alternate Protocol 1 (`Aligned.out.ss.bam`).

Protocol:

1. For generation of `intron` hints from spliced read information, please follow the first six steps in Alternate Protocol 1. The resulting file `introns.gff` contains hints from spliced RNA-Seq reads.

2. **Optional: exonpart hints**

   (a) Begin with the file `Aligned.out.ss.bam` from step 4 or step 5 in Alternate Protocol 1. Convert this file to wiggle format:

   bash input

   ```
   bam2wig Aligned.out.ss.bam > rnaseq.wig
   ```

   (b) Convert the wiggle file to hints for AUGUSTUS:

   bash input

   ```
   cat rnaseq.wig | wig2hints.pl --width=10 --margin=10 --minthresh=2 \
     --minscore=4 --prune=0.1 --src=W --type=ep \
     --UCSC=unstranded.track --radius=4.5 --pri=4 --strand="." \
     > rnaseq.gff
   ```

   When using this type of hints, remember to call AUGUSTUS with the option `--UTR=on`.

Alternate Protocol 14—Generating Hints from IsoSeq Data

Below, we describe a protocol for the identification of new mouse isoforms using single molecule Pacific Bioscience (PacBio) RNA-Seq reads. For clarity these command lines are without parallel execution. However, you can easily split the genome by chromosomes and run alignments in parallel. The tested input consisted of circular consensus sequences (ccs) that often constitute near-full length transcripts and needed not be corrected using other short read data, although we suspect this may improve mapping accuracy. In this protocol, the input genome is hard-masked, i.e. in `genome.fa`, nucleotides that are part of repetitive sequence elements are substituted by the letter `N`; however, the protocol also works with soft-masked genomes.

Required software:

- GMAP (`http://research-pub.gene.com/gmap/`)

Required files:

- Genome file in FASTA format (preferrably soft-masked).
- File with IsoSeq data in FASTA format. IsoSeq must be error-corrected, e.g. by alignment of Illumina reads. Otherwise, no (or very few) alignments will be reported in this protocol.

Protocol:

1. Prepare a GMAP index and align reads to genome:

bash input

```
mkdir gindex
gmap_build -D gindex/ -d rat genome.fa
gmap -D gindex/ -d rat isoseq.fa --min-intronlength=30 --intronlength=500000 \
  --trim-end-exons=20 -f 1 -n 0 > gmap.psl
```

The option `-n 0` to `gmap` lets it report only the alignment(s) with maximal score, no suboptimal alignments.

2. Convert the output of GMAP to hints for AUGUSTUS:

bash input

```
cat gmap.psl | sort -n -k 16,16 | sort -s -k 14,14 | perl -ne \
  '@f=split; print if ($f[0]>=100)' | blat2hints.pl --source=PB \
  --nomult --ep_cutoff=20 --in=/dev/stdin --out=isoseq.gff
```

Alternate Protocol 15—Generating Hints from Protein Data

Protein to genome alignments can aid the prediction of CDS - including the correct reading frame, start- and stop-codon positions - and the prediction of introns. Suitable hints for AUGUSTUS can be generated by various protein to genome alignment tools. In general, there are tools that produce alignments with a high specificity at the expense of sensitivity (e.g. GenomeThreader (Gremme, 2013b)) and more sensitive but less specific tools (e.g. Exonerate (Slater and Birney, 2005)). The optimal choice of alignment tools depends on the number of proteins and on the degree of relatedness of proteins and genome, and on the available computational resources (e.g. Exonerate is slow in comparison to other aligners, and will require data parallelization).

If the number of proteins to be aligned is 'small' and source proteins originate from a species that is rather closely related to the target genome species, most spliced alignment tools will produce results that serve well for gene prediction with AUGUSTUS in reasonable computational time.

However, if the degree of relatedness between source proteins and target genome is larger, you should be aware that some tools - such as GenomeThreader - will only report very reliable alignments, and other tools - such as Exonerate - may additionally report a large number of short and unspecific alignments that may not aid gene prediction. It is possible to adapt the parameters for hints integration with AUGUSTUS to both types of alignment results. Running AUGUSTUS with a huge number of hints requires memory (RAM) in a dimension that may not be justified for the task of gene prediction. It is possible to collapse redundant hints, but spurious hints may still lead to a significant memory requirement.

Generally, we recommend using an aligner with a high specificity rather than an aligner with a high sensitivity, in particular if you are aligning more than one proteome to the target genome.

Required software:

- Protein to genome alignment tool, e.g. GenomeThreader (`http://genomethreader.org/`) or Exonerate (`https://www.ebi.ac.uk/about/vertebrate-genomics/software/exonerate`)

- `startAlign.pl` and `align2hints.pl` from BRAKER (`http://bioinf.uni-greifswald.de/bioinf/braker`)

Required files:

- Genome file in FASTA format
- File with protein sequences in FASTA format

For demonstration purposes, you can use the same files as in Alternative Protocol 2 on page 13.

Protocol:

1. Align protein sequences to target genome with `startAlign.pl`. This script will call the specified alignment tool with data parallelization and custom parameters that are suitable for using produced alignments for predicting genes with AUGUSTUS. Specify parameter `--CPU=INT` for parallelization, note that `startAlign.pl` will use at most as many cores as there are genomic sequences:

   bash input

   ```
   startAlign.pl --genome=genome.fa --prot=protein.fa --prg=gth
   ```

   The option `--prg` specifies the aligner to be called, `gth` is the executable file of GenomeThreader. Alternatively, you could specify `exonerate`. If you choose to run Exonerate, we strongly recommend a multi-core machine for speeding up the alignment process.

   Above command will produce a file `align_toolname/toolname.concat.aln`, e.g. if you ran `startAlign.pl` with `--prg=gth`, the output file will be `align_gth/gth.concat.aln`.

2. Convert the alignment output to a hints file:

   bash input

   ```
   align2hints.pl --in=align_gth/gth.concat.aln --out=prot.hints --prg=gth
   ```

   Since output formats of alignment tools differ, you need to specifiy an aligner that was used for generating the output with `--prg=aligner`. `align2hints.pl` is compatible with GenomeThreader, Exonerate, Spaln, GEMOMA, and Scipio.

   By default, `align2hints.pl` will produce intron, start-codon, stop-codon, and CDSpart hints from protein alignments. CDSpart hints are hints for CDS, including the reading frame, that were trimmed for several nucleotides at both ends. Using CDSpart hints instead of exact CDS hints gives more flexibility to AUGUSTUS for predicting an agreeing gene structure.

   `align2hints.pl` has several options worth noting:

   - If you use Exonerate to produce the alignments, you should specify `--genome_file=genome.fa`, otherwise no start-codon hints will be produced.
   - `--maxintronlen=n` (default 350000) and `--minintronlen=n` (default 41) define the maximum and minimum length of intron hints. If, for example, your species is known to have longer introns, you might want to adapt the appropriate parameter accordingly.

Alternate Protocol 16—Generating Hints from ESTs and Medium Size RNA-Seq Reads

ESTs and medium size RNA-Seq reads (>∼450 nt) are suitable for generating intron, exonpart (sometimes abbreviated as "ep") and exon hints.

Required software:

- BLAT and pslCDnaFilter (`http://hgdownload.cse.ucsc.edu/admin/exe/`)

Required files:

- Genome file in FASTA format; for demonstration purposes, you can use the file `augustus/examples/autoAug/genome.fa`
- File with ESTs in FASTA format; for demonstration purposes, you can use the file `augustus/examples/autoAug/cdna.fa`

Protocol:

1. Align ESTs/cDNAs/medium sized RNA-Seq reads to genome using BLAT:

   bash input
   ```
   blat -noHead -minIdentity=92 genome.fa cdna.fa cdna.psl
   ```

2. Filter alignments to obtain those that are potentially most useful for gene prediction:

   bash input
   ```
   pslCDnaFilter -minId=0.9 -localNearBest=0.005 -ignoreNs -bestOverlap cdna.psl \
     cdna.f.psl
   ```

   This will produce a summary similar to:

   bash output
   ```
                          seqs    aligns
                 total:   2982    3243
           drop overlap:  25      41
         drop localBest:  108     199
                  kept:   2982    3003
   ```

   and the filtered alignment file `cdna.f.psl`.

3. Sort filtered alignments according to start position and target sequence name:

   bash input
   ```
   cat cdna.f.psl | sort -n -k 16,16 | sort -s -k 14,14 > cdna.fs.psl
   ```

4. Convert psl-file to hints:

bash input

```
blat2hints.pl --in=cdna.fs.psl --out=cdna.hints --minintronlen=35 --trunkSS
```

Supplemental Protocol 17 – Manual Hints

Sometimes, you want to enforce the prediction of certain gene structures, or parts of gene structures. Provide such gene structures in a hints file, specifying the source tag `M`. AUGUSTUS will try to predict genes with those exact hints unless there are no valid gene structures possible.

File contents example: `hints.gff`

```
HS04636 manual  CDSpart 500  599 .   +   .   group=gene1; source=M
HS04636 manual  intron  600  700 .   +   .   group=gene1; source=M
HS04636 manual  CDSpart 701  900 .   +   .   group=gene1; source=M
```

Basic Protocol 18—Running AUGUSTUS with Hints

Required files:

- Soft-masked genome file in FASTA format, e.g. `genome.fa`.
- Parameter files that are suitable for the target genome (see Training AUGUSTUS on training).
- One or more hints files with extrinsic evidence, e.g. `hints.gff`.

Protocol:

1. Concatenate all hints. If you have created more than one hints file, e.g. `hints.proteins.gff` from protocol Alternate Protocol 15 and `hints.rnaseq.gff` from protocol Alternate Protocol 13, then join them in a single file:

   bash input

   ```
   cat hints.proteins.gff hints.rnaseq.gff > hints.gff
   ```

   Hints from different sources can still be distinguished by `augustus` via the `src` attribute in the last column, e.g. with `src=P` and `src=E` for hints from proteins and RNA-Seq, respectively. An example of a hints file is shown in step 6 on page 11.

2. Set hint parameters. Start by making a copy of an extrinsic configuration file.

   bash input

   ```
   cp augustus/config/extrinsic/extrinsic.M.RM.E.W.P.cfg extrinsic.bug.cfg
   ```

   Here, `extrinsic.bug.cfg` is an example name of this extrinsic configuration file. It contains a relatively small number of parameters with which you can adjust how 'seriously' the hints from the different sources are taken (see step 4.). As the name *hint* suggests, AUGUSTUS is, in principle, allowed to choose gene structures that contradict some of the evidence it

is given. This is so because extrinsic evidence can also be "wrong", e.g. protein similarity to a particular region suggests that the region is protein-coding when, in fact, the region is a pseudogene or the similarity occurred by chance. Another example of "wrong" evidence is RNA-Seq from non-protein coding genes, when AUGUSTUS is set to find protein-coding genes only.

In the example above the tokens M, RM, E, W and P stand for **m**anual hints, hints from **r**epeat **m**asking, hints from **E**ST or RNA alignment, hints from transcriptome coverage ("**w**iggle" track) and hints from **p**rotein alignments, respectively. Use an appropriate template from the same folder that contains your extrinsic evidence sources. This specifically means that every token after `src=` in `hints.gff` must also be listed in `extrinsic.bug.cfg`.

In the easiest case, `extrinsic.bug.cfg` needs not be changed. However, if your hints file contains sources that are not specified in `extrinsic.bug.cfg`, or if AUGUSTUS appears to make systematic errors with regard to extrinsic evidence integration (see step 4 below), this file needs to be adjusted. Figure 5 shows an excerpt of the extrinsic config file.

3. Predict genes using hints:

bash input

```
augustus --species=bug \
   --extrinsicCfgFile=extrinsic.bug.cfg --hints file=hints.gff \
   --allow_hinted_splicesites=atac \
   --softmasking=on genome.fa > augustus.hints.gff
```

This command finds a most likely gene structure for every sequence in the genome *given* the extrinsic evidence from the hints. `--allow_hinted_splicesites=atac` allows AUGUSTUS to predict also those (rare) introns that start with AT and end with AC in addition to the GT/AG and GC/AG introns that are allowed by default. With above command, no alternative splicing is predicted. However, the option `--alternatives-from-evidence=on` makes AUGUSTUS report alternative transcripts when they are suggested by hints (Stanke et al., 2008). This means that each gene can have more than one transcript. The grouping of transcripts to genes is specified in the last column, but also apparent from the name, e.g. `g12.t3` refers to the third-best supported transcript of the twelfth gene.

4. **Optional:** Adjust hint parameters.

The example extrinsic configuration files were adjusted to work well on what we believe to be typical hint datasets. However, this mechanism to integrate extrinsic evidence is general and the parameters cannot be universally optimal. We therefore recommend loading the predictions from step 3 into a genome browser as described in step 2 of Alternate Protocol 4, together with tracks for the evidence, like alignments that were used to generate hints. For this purpose, it is sufficient that this is done on a sample of the genome in order to visually inspect, say, a couple of dozen genes. Potential systematic errors in the hint generation could become apparent; for example, if RNA-Seq was aligned to a different assembly version and thus there appears to be no clear resemblance between the positions where AUGUSTUS predicts introns and where reads are spliced. Or, "undermasking" leads to large numbers of regions with protein alignments that can not actually be protein-coding genes. Moreover, the visual inspection can bring to light whether one of the two problems that can happen in hint integration is clearly prevailing: i) the AUGUSTUS genes disagree with hints where they

should follow them or ii) the hints are wrong but still incorporated by AUGUSTUS into a gene structure. An example of the first problem could be that many introns suggested by trustable RNA-Seq alignments are not incorporated into a gene structure although biologically valid compatible gene structures with these introns exist. In this case, the intron bonus for the RNA-Seq source could be increased (e.g. by increasing 1e6 to 1e9 in `extrinsic.bug.cfg`, cf. Figure 5). An example of the second problem could be that for the protein alignment a too lax similarity threshold was used and – as a result – many protein homology hints suggest that regions are coding although they are actually not. In this case, the CDSpart (and CDS) bonus could be decreased, say from 1e5 to 1e2).

Tip: Do not be timid to first make a large change of the parameters in the right direction (increasing or decreasing). The predictions are not very sensitive to the hint parameters and if the effect is that the respective other problem becomes prevalent, then an intermediate parameter value can still be searched.

It should be noted that, although it is preferable that this manual correction step is not required (and it often can be omitted), manual inspection cannot be automatized away in general at this point. This is so because extrinsic evidence sources are kept general and their distribution may therefore in principle be grossly different on other data sets. For example, if hints were generated from an existing highly complete and accurate reference annotation of coding sequences in order to let AUGUSTUS extend UTRs to the CDS exons, then one may want to choose a very small (and therefore effective) CDS malus in order to prevent AUGUSTUS from predicting additional coding exons.

Supplemental Protocol 19—Joining Several Predictions

In principle, you can integrate different types of evidence in a single AUGUSTUS run. However, sometimes you encounter scenarios where it makes sense to join several gene sets for a particular genome, i.e. gene sets that were produced by AUGUSTUS runs with different parameter sets. The tool `joingenes` was developed to join such different gene sets into one gene set.

Required files:

- Two or more files with gene predictions in GTF format.

Protocol:

1. Join two gtf files with different gene sets with `joingenes`:

   bash input

   ```
   joingenes --genesets=set1.gtf,set2.gtf,... --output=joined.gtf
   ```

   If the gene sets are of different confidence, the option `--priorities=pr1,pr2,...` allows prioritizing the different sets (`pr1,pr2,...` are positive integers).

Supplemental Protocol 20—Data Parallelization

The gene prediction problem for a large genome can be broken into smaller tasks that can be executed in parallel on a multi-core machine or a cluster. Be aware that there are two common

scenarios that require slightly different data handling: A genome file may contain few but very long chromosome-style sequences (follow Suppl. Protocol 20), or a genome file may contain many short contig- or scaffold-style sequences (follow Suppl. Protocol 21).

Required files:

- A genome file in FASTA format, e.g. `genome.fa`
- If applicable: a file with hints for AUGUSTUS, e.g. `hints.gff`

Protocol:

1. If you have a genome file with few long sequences, split the file in a way that each smaller file contains exactly one sequence:

   bash input

   ```
   mkdir split
   splitMfasta.pl genome.fa --outputpath=split
   ```

2. Rename the split single FASTA files according to their FASTA names:

   bash input

   ```
   for f in split/genome.split.*;
   do
     NAME=‘grep ">" $f‘; mv $f ${NAME#>}.fa
   done
   ```

3. Next, we need to obtain the length of each genomic sequence. To this end, run `summarizeATGCcontent.pl`:

   bash input

   ```
   summarizeACGTcontent.pl genome.fa > summary.out
   ```

   It produces an output similar to this:

   File contents example: `summary.out`

   ```
   15072423 bases. chrI   BASE COUNT 4835938 a   2695881 c  2692152 g   4848452 t
   15279345 bases. chrII  BASE COUNT 4878197 a   2769219 c  2762213 g   4869716 t
   13783700 bases. chrIII BASE COUNT 4444651 a   2449148 c  2466334 g   4423567 t
   17493793 bases. chrIV  BASE COUNT 5711039 a   3034770 c  3017014 g   5730970 t
   13794 bases.    chrM   BASE COUNT    4335 a      1225 c     2055 g      6179 t
   20924149 bases. chrV   BASE COUNT 6750394 a   3712061 c  3701398 g   6760296 t
   17718866 bases. chrX   BASE COUNT 5747199 a   3119708 c  3117874 g   5734085 t
   summary: BASE COUNT     32371753 a   17782012 c  17759040 g   32373265 t
   total 100286070bp
   gc: 0.35439669736784%
   ```

4. The sequence file names and lengths are the basis of a file `chr.lst` that will later be used to create the actual AUGUSTUS job scripts. There are two different scenarios to be considered:

   - ***Ab initio* prediction:**

bash input

```
grep "bases" summary.out | \
  perl -pe 's/^(\d+) bases.\s+(\S+) BASE.*/split/$2\.fa\t1\t$1/;' \
  > chr.lst
```

The resulting file should look like this:

File contents example: chr.lst

```
split/chrI.fa     1       15072423
split/chrII.fa    1       15279345
```

This tabular separated file in the first column contains a FASTA file name that contains exactly one sequence, the number 1 in the second column (prediction start) in the second column and the length of the sequence in the last column (prediction end).

- **Prediction with a hints file:**

bash input

```
grep "bases" summary.out | perl -pe \
  's/^(\d+) bases.\s+(\S+) BASE.*/split/$2\.fa\thints.gff\t1\t$1/;' \
  > chr.lst
```

The file will then contain a hints file in the second column and shift all other columns to the right:

File contents example: chr.lst

```
split/chrI.fa   hints.gff  1       15072423
split/chrII.fa  hints.gff  1       15279345
```

5. Creating AUGUSTUS jobs. Let `$augDir` be the directory to which you want to write your temporary gene predictions for genome chunks; and let `$augCall` be the AUGUSTUS command (e.g. `augustus --UTR=on --print_utr=on --AUGUSTUS_CONFIG_PATH=somePath/config --exonnames=on --codingseq=on --species=yourspecies`). Furthermore, it can be helpful for job monitoring to assign a job script prefix, e.g. `$myPrefix_`. Again, we consider both scenarios:

   - ***Ab initio* prediction** (and prediction with hints where the hints file is so small that you do not want to partition it):

bash input

```
createAugustusJoblist.pl --sequences=chr.lst --wrap="#" --overlap=5000 \
  --chunksize=1252500 --outputdir=$augDir/ --joblist=jobs.lst \
  --jobprefix=$myPrefix_ --command "$augCall"
```

   - **Prediction with a huge hints file.** If the genome is large, the corresponding hints may be large, too. It may be helpful for speeding up job processing if AUGUSTUS does not have to read all hints for every single subtask (However, AUGUSTUS filters

out the hints that are not applicable to the input sequence or range.). The option `--partitionHints` will split the hints file into single batches that correspond to each input sequence file:

bash input

```
createAugustusJoblist.pl --sequences=chr.lst --wrap="#" --overlap=100000 \
  --chunksize=1100000 --outputdir=$augDir/ --joblist=jobs.lst \
  --jobprefix=$myPrefix_ --partitionHints --command "$augCall"
```

`createAugustusJoblist.pl` creates a number of AUGUSTUS job scripts for sequence segments. The segments overlap, here by 100000 bp. This overlap length should be set at least in the order of the length of a long gene of that species. Otherwise it may happen that a gene is not completely contained in any of the (overlapping) segments. The job scripts are named `$myPrefix_$i` (where `$i` is the i^{th} job). The scripts are listed in file `jobs.lst` (assume that `$myPrefix_` has the value `aug`):

File contents example: `jobs.lst`

```
aug_1
aug_2
aug_3
(...)
```

6. Run subtasks in parallel on a multi-core machine or cluster. There are many ways to achieve this. Here, we illustrate two options:

   (a) **Multi-core machine without grid engine.** GNU `parallel` processes a list of jobs in parallel on a single machine. In the directory with all the sub task scripts, run:

   bash input

   ```
   parallel -j 8 --bar --no-notice "nice ./{}" < jobs.lst
   ```

   (Replace 8 by the numbers of cores that you would like to occupy on your machine.)

   (b) **Cluster (or similar) with Portable Batch System (PBS).** For executing on a system such as PBS, you want to determine the number of tasks first, e.g. by counting the number lines in `jobs.lst`:

   bash input

   ```
   wc -l jobs.lst
   ```

   bash output

   ```
   30
   ```

   Next, you write a PBS submission script (custom tailored to your cluster's requirements, of course); insert the number of tasks as an end variable of the array (e.g. `#PBS -t 1-30`, 30 is the number of tasks, here). It could look like this:

File contents example: jobs.sh

```
#!/bin/sh
#PBS -N jobname
#PBS -l walltime=100:00:00
#PBS -t 1-30

/absolute/path/to/$myPrefix_${PBS_ARRAYID}
```

Make sure that the execution node knows where to find the job (e.g. by specifying an absolute path to the job script). Submit the script, e.g. with:

bash input

```
qsub jobs.sh
```

7. After all jobs have finished, have a look at the error files in the directory `$augDir/`:

   bash input

   ```
   ls -lh $augDir/*.err
   ```

   Ideally, none of the jobs produced an error and all files are empty (show the value 0 in column 5 of the output of `ls`). If you find error files with contents, check what went wrong before proceeding.

8. Next, the gff output of all sub tasks will be merged. For this, we need files in the correct order along sequences. One way to achieve this is to filter all output file names from the job scripts:

   bash input

   ```
   for x in ‘cat jobs.lst‘
   do
   cat $x | perl -ne ’if(m/--outfile=(\S*) --errfile/){print $1} | \
     join_aug_pred.pl > augustus.gff
   done
   ```

   `join_aug_pred.pl` cleans the redundancies that may occur from duplicated predictions in overlapping regions. It also renames the genes sequentially so that the gene names become globally unique. It requires that the input is given in the order such that neighboring regions are input sequentially as is achieved by above command. It produces an output file, here `augustus.gff`, that is comparable to the output of running AUGUSTUS without any data parallelization.

Supplemental Protocol 21—Parallelization for Short Sequences

1. If you have a file with very many sequences, e.g. thousands of sequences, a solution with one file per sequence may be inefficient. In this case, split the genome into smaller files that contain a minimum amount of nucleotides, each:

bash input

```
mkdir split
splitMfasta.pl genome.fa --outputpath=split --minsize=1000000
```

2. Determine the number of split genome files, e.g. run:

bash input

```
ls split/genome.split.*.fa | wc -l
```

The output could be this:

bash output

```
30
```

3. **Optional: Split hints file.** If the hints file is large, split it into smaller files that contain exactly those hints that correspond to a split genome file. First, create a list of sequence names of each split genome file:

bash input

```
for ((i=1; i<=30; i++));
do
  fgrep ">" split/genome.split.$i.fa | perl -pe 's/^>//' > part.$i.lst
done
```

File contents example: `part.1.lst`

```
seq1
seq2
seq3
```

Next, identify hints that correspond to the genomic files. This could in principle be accomplished by a `grep`, but `getLinesMatching.pl` is more efficient:

bash input

```
mkdir split_hints
for ((i=1; i<=30; i++));
do
  cat hints.gff | getLinesMatching.pl part.$i.lst 1 > split_hints/hints.split.$i.gff
done
```

4. Execute all subtasks on a multi-core machine or cluster.

   (a) **Multi-core machine without grid engine** with GNU `parallel` (modify the AUGUSTUS call to your requirements):

bash input

```
seq 1 30 | parallel -j 8 --bar --no-notice "nice augustus --UTR=on \
  --print_utr=on --alternatives_from_evidence=1 --extrinsicCfgFile=someFile.cfg \
  --AUGUSTUS_CONFIG_PATH=somePath/config --exonnames=on --codingseq=on \
  --species=yourspecies --hintsfile=split_hints/hints.split.{}.gff \
  split/genome.split.{}.fa > $augDir/augustus.{}.out \
  2> $augDir/augustus.{}.err"
```

(b) **Cluster (or similar) with PBS.** Prepare a submission script, e.g.:

File contents example: `jobs.sh`

```
#!/bin/sh
#PBS -N jobname
#PBS -l walltime=100:00:00
#PBS -t 1-30

augustus --UTR=on --print_utr=on --alternatives_from_evidence=1 \
  --extrinsicCfgFile=someFile.cfg --AUGUSTUS_CONFIG_PATH=somePath/config \
  --exonnames=on --codingseq=on --species=yourspecies \
  --hintsfile=split_hints/hints.split.${PBS_ARRAYID}.gff \
  split/genome.split.${PBS_ARRAYID}.fa > $augDir/augustus.${PBS_ARRAYID}.out \
  2> $augDir/augustus.${PBS_ARRAYID}.err
```

Submit the script, e.g. with:

bash input

```
qsub jobs.sh
```

5. After all tasks have finished, check for errors as described in step 7 of the *Basic Protocol*.
6. Merge the AUGUSTUS output:

bash input

```
for (i=1; i<=30; i++);
do
  cat $augDir/augustus.$i.out | join_aug_pred.pl > augustus.gff
done
```

Guidelines for Understanding Results

Genome Browser

We strongly recommend to visualize the predicted genes along evidence on the gene structures in a genome browser, such as the UCSC genome browser (Casper et al., 2017), JBrowse (Skinner et al., 2009) or Artemis (Carver et al., 2011). If AUGUSTUS was run with several settings, each prediction can be a 'track' on the browser. Other tracks could be for RNA-Seq alignments, for repeats, for protein homology and for other gene sets, e.g. previously existing ones or from other structural

genome annotation software. A short example, how the UCSC genome browser can be used to display gene tracks is given in step 2 of protocol 7.2. However, for comprehensive instructions we refer the reader to the chapter on the UCSC Genome Browser in a previous issue in the series *Current Protocols in Bioinformatics* (Karolchik et al., 2012).

Number of Genes

There are several reasons to be cautious before using the number of predicted genes as an overall estimate of the number of genes in the genome. One reason is, that a fractioned assembly may lead to a siginicant proportion of genes that are truncated by the scaffold boundary. In the AUGUSTUS numbering of genes such partial genes are counted as well and thus the number of (complete) genes may be smaller than the overall number of reported genes. The genes with complete coding sequence can be identified by searching for all transcripts, which have a predicted start *and* stop codon. An estimate of the number of genes that the fragmented genes represent could be obtained indirectly through their total coding sequence length, if one assumes that the partial genes have on average the same coding sequence length as the complete ones and that the assembly is at least near-complete. Another reason for a larger deviation from the number of predicted and actual genes can be what we call *split gene* or *joined gene* error. A split gene error occurs when a single actual gene is predicted as several genes and results in an overestimate of the number of genes. On the other hand, a joined gene error occurrs when two or more genes are predicted as one gene and results in an underestimate of the number of genes. It should be noted that above gene number deviations can even occur if the number of predicted exons and the number of actual exons are similar, as is frequently the case. Often, the inspection of individual suspicious gene structures using BLAST can reveal a systematic tendency in either error.

Bad Results

The accuracy statistics of *ab initio* AUGUSTUS reported on an independent test set of gene structures gives an indication of the relative success of the training procedure. A successful example is given in step 7 of section 7.7.1. The gene level accuracy can be quite low, even below 0.2, if there is on average a large number of introns per gene and if those introns have a high variance of intron length, as is the case in vertebrate genomes. This does not necessarily mean that the results are unusually inaccurate. However, an exon level accuracy that is significantly below 0.7 would suggest that something went wrong with the training. The cause of the problem could be a low assembly quality, undetected repeats that have a sequence content similar to genes, a high error rate in the training and test set or, in rare cases, unusual properties of the gene structures like a nonstandard set of stop codons. In this case, we recommend to investigate these possible causes before proceeding with evidence-based gene prediction. It is true that the predictions using hints are likely to contain fewer errors than the *ab initio* predictions. However, they could then likely be better if a remedy was applied in the first place.

Commentary

Background Information

AUGUSTUS belongs to a class of genome annotation tools that can be described as *statistical gene finding*, because probabilistic models are used at least in part to identify biological signals and the gene structures. Other examples of such programs are GeneMark (`http://opal.biology.gatech.edu/GeneMark/`), SNAP (`http://github.com/KorfLab/SNAP`) and Fgenesh (`http://www.softberry.com`). Another class of tools for genome annotation are *transcriptome reconstruction* tools, which rely on RNA-Seq or other transcriptome sequencing data and can use the genome as guide for assembling transcripts as well, e.g. StringTie (`https://ccb.jhu.edu/software/stringtie/`). A third popular class of tools for genome annotation are methods based on alignments of amino acid sequences of previously annotated *homologs*, e.g. GenomeThreader (`http://genomethreader.org`) and Exonerate (`https://www.ebi.ac.uk/about/vertebrate-genomics/software/exonerate`). Here, the boundaries to statistical gene finding are less clear, as in presence of only remote homology similarity finding becomes a probabilistic weighing and signal models are required to find exon boundaries. Transcriptome reconstruction tools and most protein alignment tools do not use clade-specific statistical models and therefore do not require training. On the other hand, they cannot be exclusively used to obtain a whole-genome annotation, as they are by design unable to find genes or parts of genes that are not represented in the data. The RNA-seq Genome Annotation Assessment Project (rGASP) provided a comparison of several commonly used tools of the first two above classes and the overall accuracy in the task of structurally annotating protein-coding genes tended to be higher for statistical gene finders (Supplements of (Steijger et al., 2013)).

Troubleshooting

The AUGUSTUS web site at `http://bioinf.uni-greifswald.de/augustus/` contains a link to questions and answers, to further documentation and tutorials and a contact email address. AUGUSTUS is under ongoing development and new features are being developed. In fact, at the time of writing this chapter a grant project has just started to improve the usability of AUGUSTUS. From time to time users find a bug, e.g. `augustus` crashes on certain input. In such a case you can help the development if you provide a small example and the command lines so the developers can first reproduce the problem and eventually fix it.

Acknowledgement

This chapter is based on research that was funded partially by Deutsche Forschungsgemeinschaft grants STA 1009/6-1 and STA 1009/10-1. We thank Dr. Malte Wellnitz and Matthis Ebel for proofreading.

Literature Cited

Carver, T., Harris, S. R., Berriman, M., Parkhill, J., and McQuillan, J. A. (2011). Artemis: an integrated platform for visualization and analysis of high-throughput sequence-based experimental data. *Bioinformatics*, 28(4):464–469.

Casper, J., Zweig, A. S., Villarreal, C., Tyner, C., Speir, M. L., Rosenbloom, K. R., Raney, B. J., Lee, C. M., Lee, B. T., Karolchik, D., et al. (2017). The ucsc genome browser database: 2018 update. *Nucleic acids research*, 46(D1):D762–D769.

Castellana, N., Payne, S., Shen, Z., Stanke, M., Bafna, V., and Briggs, S. (2008). Discovery and revision of *Arabidopsis* genes by proteogenomics. *Proceedings of the National Academy of Sciences USA*, 105(52):21034–21038.

Chen, N. (2004). Using repeatmasker to identify repetitive elemnts in genomic sequences. *Current protocols in bioinformatics*, 5(1):4.10. 1–4.10. 14.

Daehwan, K., Pertea, G., Trapnell, C., Pimentel, H., Kelley, R., and Salzberg, S. (2013). TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, 14:R36.

Dobin, A., Davis, C., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., Batut, P., Chaisson, M., and Gingeras, T. (2013). STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21.

Gremme, G. (2013a). Computational gene structure prediction.

Gremme, G. (2013b). *Computational Gene Structure Prediction*. PhD thesis, Universität Hamburg.

Haas, B. J., Delcher, A. L., Mount, S. M., Wortman, J. R., Smith Jr, R. K., Hannick, L. I., Maiti, R., Ronning, C. M., Rusch, D. B., Town, C. D., et al. (2003). Improving the arabidopsis genome annotation using maximal transcript alignment assemblies. *Nucleic acids research*, 31(19):5654–5666.

Hoff, K. and Stanke, M. (2013). WebAUGUSTUS – a web service for training AUGUSTUS and predicting genes in eukaryotes. *Nucleic Acids Research*.

Karolchik, D., Hinrichs, A. S., and Kent, W. J. (2012). The ucsc genome browser. *Current protocols in bioinformatics*, 40(1):1–4.

Keller, O., Odronitz, F., Stanke, M., Kollmar, M., and Waack, S. (2008). Scipio: Using protein sequences to determine the precise exon/intron structures of genes and their orthologs in closely related species. *BMC Bioinformatics*, 9(1):278.

Kent, W. (2002). BLAT — The BLAST-Like Alignment Tool. *Genome Research*, 12(4):656–664.

König, S., Romoth, L., Gerischer, L., and Stanke, M. (2016). Simultaneous gene finding in multiple genomes. *Bioinformatics*, 32(22):3388–3395.

Lomsadze, A., Burns, P., and Borodovsky, M. (2014). Integration of mapped RNA-Seq reads into automatic training of eukaryotic gene finding algorithm. *Nucleic Acids Research*, 42(15):e119.

Noble, W. S. (2009). A quick guide to organizing computational biology projects. *PLOS Computational Biology*, 5(7):1–5.

Pirovano, W., Boetzer, M., Derks, M., and Smit, S. (2015). NCBI-compliant genome submissions: tips and tricks to save time and money. *Briefings in Bioinformatics*, page bbv104.

Price, A. L., Jones, N. C., and Pevzner, P. A. (2005). De novo identification of repeat families in large genomes. *Bioinformatics*, 21(suppl_1):i351–i358.

Skinner, M., Uzilov, A., Stein, L., Mungall, C., and Holmes, I. (2009). JBrowse: A next-generation genome browser. *Genome Research*, 19:1630–1638.

Slater, G. and Birney, E. (2005). Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics*, 6(1):31.

Stanke, M. and Borodovsky, M. (2018). *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, chapter Genome Structural Annotation. Wiley. in preparation.

Stanke, M., Diekhans, M., Baertsch, R., and Haussler, D. (2008). Using native and syntenically mapped cDNA alignments to improve *de novo* gene finding. *Bioinformatics*, 24(5):637–644.

Stanke, M., Keller, O., Gunduz, I., Hayes, A., Waack, S., and Morgenstern, B. (2006a). AUGUSTUS: ab initio prediction of alternative transcripts. *Nucleic Acids Res. 3*, 34:W435–W439.

Stanke, M., Schöffmann, O., Dahms, S., Morgenstern, B., and Waack, S. (2006b). Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC Bioinformatics*, 7:62.

Stanke, M., Steinkamp, R., Waack, S., and Morgenstern, B. (2004). AUGUSTUS: a web server for gene finding in eukaryotes. *Nucleic Acids Res.*, 32:W309–W312.

Steijger, T., Abril, J., Engstrom, P., Kokocinski, F., Akerman, M., Alioto, T., Ambrosini, G., Antonarakis, S., Behr, J., Bohnert, R., Bucher, P., Cloonan, N., Derrien, T., Djebali, S., Du, J., Dudoit, S., Gerstein, M., Gingeras, T., Gonzalez, D., Grimmond, S., Habegger, L., Iseli, C., Jean, G., Kahles, A., Lagarde, J., Leng, J., Lefebvre, G., Lewis, S., Mortazavi, A., Niermann, P., Rätsch, G., Reymond, A., Ribeca, P., Richard, H., Rougemont, J., Rozowsky, J., Sammeth, M., Sboner, A., Schulz, M., Searle, S., Solorzano, N., Solovyev, V., Stanke, M., Steijger, T., Stevenson, B., Stockinger, H., Valsesia, A., Weese, D., White, S., Wold, B., Wu, J., Wu, T., Zeller, G., Zerbino, D., Zhang, M., Hubbard, T., Guigo, R., Harrow, J., and Bertone, P. (2013). Assessment of transcript reconstruction methods for RNA-seq. *Nat Meth*, 10(12):1177–1184. Analysis.

Wu, T. and Nacu, S. (2010). Fast and snp-tolerant detection of complex variants and splicing in short reads. *Bioinformatics*, 26:873–881.

Wu, T. D. and Watanabe, C. K. (2005). GMAP: a genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics*, 21:1859–1875.

Tables

	exon	
A	sensitivity	specificity
zebrafish	80.2%	71.0%
chicken	73.6%	66.1%
human	71.8%	59.3%

Table 1: *Ab initio* accuracy after cross-species training. Genes were predicted on species B = zebrafish, after training on species A. Accuracy was measured for predicting coding exons (CDS). Some loss in accuracy from cross-training from mammal or bird to fish is apparent. However, even between these distant clades foreign parameters are not useless. Sensitivity = true positives / actual positives, specificity = true positives / predicted positives.

Translation table number	a a a a a a a a a a a a a a a a c c c c c c c c c c c c c c c c g g g g g g g g g g g g g g g g t t t t t t t t t t t t t t t t
	a a a a c c c c g g g g t t t t a a a a c c c c g g g g t t t t a a a a c c c c g g g g t t t t a a a a c c c c g g g g t t t t
	a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t a c g t
1	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S * C W C L F L F
2	K N K N T T T T * S * S M I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S W C W C L F L F
3	K N K N T T T T R S R S M I M I Q H Q H P P P P R R R R T T T T E D E D A A A A G G G G V V V V * Y * Y S S S S W C W C L F L F
4	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S W C W C L F L F
5	K N K N T T T T S S S S M I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S W C W C L F L F
6	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V Q Y Q Y S S S S * C W C L F L F
9	N N K N T T T T S S S S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S W C W C L F L F
10	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S C C W C L F L F
11	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S * C W C L F L F
12	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L S L E D E D A A A A G G G G V V V V * Y * Y S S S S * C W C L F L F
13	K N K N T T T T G S G S M I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S W C W C L F L F
14	N N K N T T T T S S S S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V Y Y * Y S S S S W C W C L F L F
15	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y Q Y S S S S * C W C L F L F
16	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y L Y S S S S * C W C L F L F
21	N N K N T T T T S S S S M I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S W C W C L F L F
22	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y L Y * S S S * C W C L F L F
23	K N K N T T T T R S R S I I M I Q H Q H P P P P R R R R L L L L E D E D A A A A G G G G V V V V * Y * Y S S S S * C W C * F L F

Table 2: Translation tables for AUGUSTUS (default is number 1). In case your species' code is not covered by this table send us a note with the string of 64 one-letter amino acid codes in the codon order in this table.

		alternatives Option			
		1	2	3	
	no	0.2	0.08	0.08	`minexonintronprob`
	alter-	0.5	0.4	0.3	`minmeanexonintronprob`
	natives	2	3	20	`maxtracks`
number of genes	227	187	187	183	
number of transcripts	227	221	248	437	

Table 3: The lower the thresholds `minexonintronprob` and `minmeanexonintronprob` and the higher `maxtracks`, the more alternative transcripts are predicted by AUGUSTUS. The column "no alternatives" shows the number of transcripts predicted in the example of the basic protocol above.

Figures

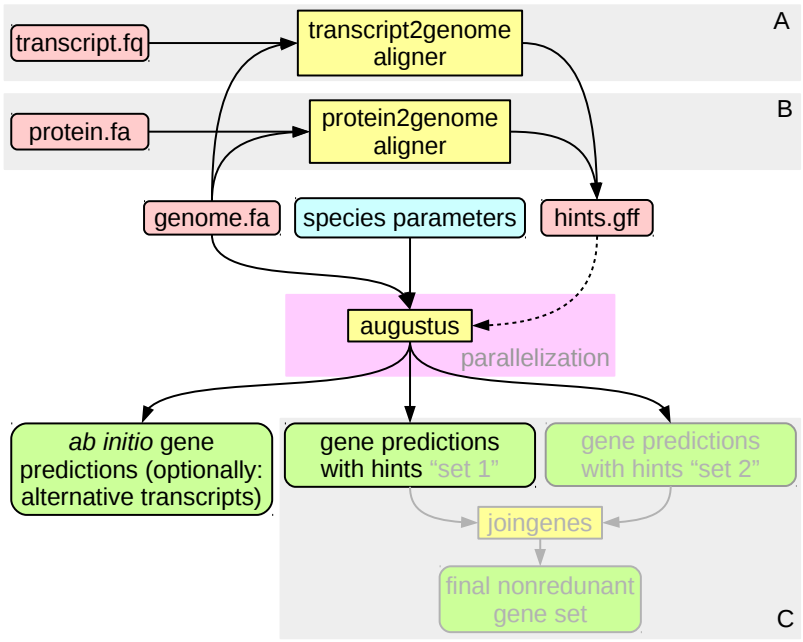

Figure 1: AUGUSTUS can be run with input of a genome sequence only, making predictions *ab initio*, or with extrinsic evidence in form of a hints file. In principle, transcript evidence (A) provides information about the location of exons. Protein evidence (B) additionally provides information about the location of coding sequences, including reading frame, and gene start and end positions. `augustus` has no parallel execution mode, but data parallelization is possible. In case several "competing" gene sets were generated for a target genome, we provide `joingenes` for merging them into a final gene set (C).

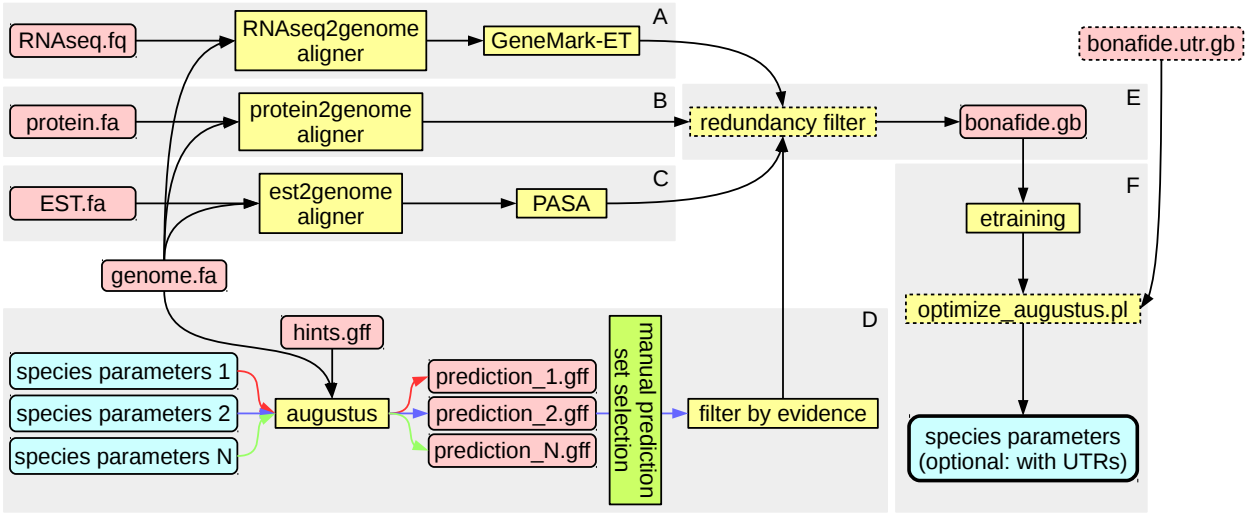

Figure 2: Illustration of workflows for training AUGUSTUS that are described in this protocol. (A) generating training genes from RNA-Seq to genome alignments and subsequent application of self-training GeneMark-ET (Lomsadze et al., 2014), (B) generating training genes from protein to genome alignments, (C) generating training genes from EST/cDNA/long-454-read to genome alignments that are converted to gene structures by transcript assembler PASA (Haas et al., 2003), (D) generating training genes via iterative prediction and filtering based on a previously existing AUGUSTUS parameter set and extrinsic evidence. (E) Regardless the source of training genes, they should preferably be cleared of redundancies. (F) Training is performed by `etraining`. Species specific parameters can optionally be refined by `optimize_augustus.pl`. In case gene models with untranslated regions (UTRs) are available, this information can also be taken into account. The aim of training AUGUSTUS is to produce a set of species-specific parameters for subsequently applying AUGUSTUS to gene prediction in a target genome.

File contents example: `bug_parameters.cfg`

```
#
# bug parameters.
#
...

stopCodonExcludedFromCDS false
# gff output options:
protein              on    # output predicted protein sequence
codingseq            off   # output the coding sequence
...
sample                       100   # the number of sampling iterations
alternatives-from-sampling   false # output alternative suboptimal transcripts
alternatives-from-evidence   false # output alternative transcripts based on explicit
                                   # evidence from hints
minexonintronprob            0.08  # minimal posterior probability of all (coding) exons
minmeanexonintronprob        0.4   # minimal geometric mean of the posterior probs of introns
                                   # and exons
maxtracks                    -1    # maximum number of reported transcripts per gene
                                   # (-1: no limit)

# global constants
# ----------------------------

/Constant/trans_init_window           20
/Constant/ass_upwindow_size           30
/Constant/ass_start                   3
/Constant/ass_end                     2
...
```

Figure 3: An excerpt of the configuration file for a species "bug". This file contains settings that can also be done (or overridden) on the command line, like `--protein=off` as well as meta parameters for training that can be automatically set by `optimize_augustus.pl` like `/Constant/ass_start`, the number of bases upstream of the consensus dinucleotide at the acceptor splice site that is considered in the pattern recognition model.

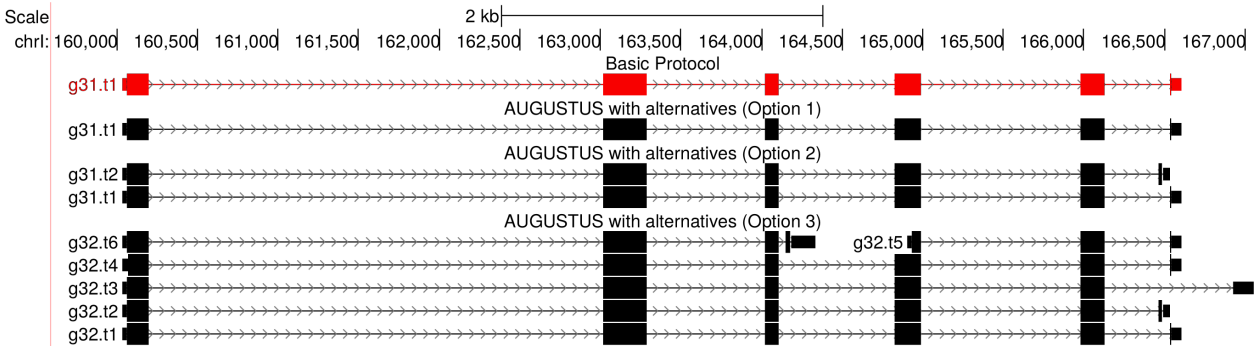

Figure 4: Alternatives from sampling. The red track shows the prediction of the basic protocol which always predicts a single transcript for each gene. The three black tracks show predictions from the Alternative protocol with increasing average number of transcripts per gene.

```
[SOURCES]
M RM E W P

[GENERAL]
      start   1        0.8  M 1  1e+100  RM  1 1    E 1   1   W 1  1    P 1  1e3
       stop   1        0.8  M 1  1e+100  RM  1 1    E 1   1   W 1  1    P 1  1e3
   exonpart   1  .992 .985  M 1  1e+100  RM  1 1    E 1   1   W 1  1.02 P 1    1
       exon   1        0.9  M 1  1e+100  RM  1 1    E 1   1   W 1  1    P 1  1e4
 intronpart   1          1  M 1  1e+100  RM  1 1    E 1   1   W 1  1    P 1    1
     intron   1        .34  M 1  1e+100  RM  1 1    E 1 1e6   W 1  1    P 1  100
    CDSpart   1     1 .985  M 1  1e+100  RM  1 1    E 1   1   W 1  1    P 1  1e5
        CDS   1          1  M 1  1e+100  RM  1 1    E 1   1   W 1  1    P 1    1
nonexonpart   1          1  M 1  1e+100  RM  1 1.15 E 1   1   W 1  1    P 1    1
```

Figure 5: An excerpt of an extrinsic configuration file. In this example, each number to the right of the column filled with `M`s that is different from 1 specifies a *bonus*. A bonus is a relative factor that the unnormalized joint probability of gene structure candidate gets for being compatible with a hint of that type and source. For example, the blue `1e6` in the intron row after the source letter E means that for each intron hint with source tag E (src=E), gene structures that have an intron with both boundaries given as in the hint are rewarded by a factor of 10^6 relatively to gene structures disregarding the intron hint. A high bonus has the effect that many of the respective hints are respected by AUGUSTUS. The green 1.15 in the nonexonpart row after the tag RM (repeat masking) specifies that for each nonexonpart hint, every gene structure gets a relative bonus factor of 1.15 *for each base* that is not an exon and not in a repeat. This discourages – but does not exclude – the overlap of exons and repeats. Repeat masking evidence can be given explicitly with hints of source RM or implicitly with a soft-masked genome and the option `softmasking` turned on. The number(s) immediately to the left of the M column other than 1 specifies a penalty (malus) for gene structures with unsupported features. For example, the red .34 in the intron row means that every intron candidate that has no intron hints supporting it is penalized by multiplying its unnormalized probability with the factor 0.34. If you decrease this number even more (say from .3 to .001) then fewer introns unsupported by hints should be predicted. This would likely decrease the false positive intron rate, but also more true unsupported introns would be missed. For more information see the file `config/extrinsic/extrinsic.cfg`.