BUSCO: assessing genome assembly and annotation completeness

Seppey, Mathieu; Manni, Mose; Zdobnov, Evgeny

How to cite

SEPPEY, Mathieu, MANNI, Mose, ZDOBNOV, Evgeny. BUSCO: assessing genome assembly and annotation completeness. In: Methods in Molecular Biology, 2019, vol. 1962, p. 227–245. doi: 10.1007/978-1-4939-9173-0_14

This publication URL: https://archive-ouverte.unige.ch/unige:123167
Publication DOI: 10.1007/978-1-4939-9173-0_14

BUSCO: Assessing Genome Assembly and Annotation Completeness

Running Head: BUSCO: Completeness Assessment

Mathieu Seppey*

Mosè Manni*

Evgeny M. Zdobnov*

*Department of Genetic Medicine and Development, University of Geneva Medical School and Swiss Institute of Bioinformatics, Geneva, Switzerland

Corresponding author: E-mail: evgeny.zdobnov@unige.ch

Abstract

Genomics drives the current progress in molecular biology, generating unprecedented volumes of data. The scientific value of these sequences depends on the ability to evaluate their completeness using a biologically meaningful approach. Here, we describe the use of the BUSCO tool suite to assess the completeness of genomes, gene sets, and transcriptomes, using their gene content as a complementary method to common technical metrics. The chapter introduces the concept of universal single copy genes, which underlies the BUSCO methodology, covers the basic requirements to set up the tool, and provides guidelines to properly design the analyses, run the assessments, and interpret and utilize the results.

Key words

BUSCO, orthologs, genome completeness, quality assessment, gene content, phylogenomics.

1. Completeness assessment

The ever-increasing volumes of sequencing data play a crucial role in advancing biological research. However, comprehensive and unbiased genomic analyses rely on the quality and completeness of such resources. This makes thorough quality control of sequencing data "products" such as genomes, genes, or transcriptomes ever more important. The assembly of reads from high-throughput sequencing technologies is a challenging procedure both theoretically and practically, especially for large genomes. Fast and accurate quality assessment of the resulting assemblies allows researchers to iteratively tweak workflows and parameters to achieve the best results. However, such evaluations are often complicated, and pose challenges for the scalability of methods, as well as for the interpretation and presentation of results.

When assessing the quality and completeness of an assembly, different complementary metrics can be used. A first step for identifying potential problems in sequencing data that are likely to hamper the

quality of the assembly, is by analyzing the k-mer distributions of reads from which the researcher can estimate sequencing bias, coverage depth, repeat content and heterozygosity of the sample. Tools such as GenomeScope [1] and KmerGenie [2] provide an easy solution for obtaining various statistics from the k-mer distributions of short reads. With a first draft assembly in hand, one can compare its size with the expected genome size estimated with experimental procedures such as by flow cytometry, or with computational methods by analyzing the k-mer distributions. Metrics such as fragment (contig/scaffold) length distributions and contig/scaffold N50, which reflect the contiguity of the assembly, are informative measures but can also be misleading. N50 value indicates that half the genome is assembled on contigs/scaffolds of length N50 or longer. Novel metrics that provide more realistic estimates of genome fragmentation have been proposed, for example, REAPR [3] provides a "corrected" N50 metric based on the identification of assembly errors. By calculating the fraction of reads that map onto the assembly, one can measure how well the original reads are represented in the genome, and the analysis of read depth can be used to identify assembly "artifacts" such as duplicated or collapsed regions.

Although the above-mentioned technical statistics are essential to estimate the overall completeness of an assembly, including intergenic regions, such measures ignore biological aspects and the important question of genomic data quality in terms of completeness of gene content. This is a crucial consideration that also affects data interpretation and helps to guide improved assembly and annotation strategies. In cases where extensive transcriptomic resources (EST, RNA-seq) for the species of interest are available, one could assess the comprehensiveness of the gene set by aligning transcripts to the assembly to obtain the proportion of mapping transcripts. However, aligning spliced transcripts to corresponding genomic loci can be problematic, and results highly depend on the tools and parameters used for mapping.

An attractive alternative, which complements the strategies described above, is to quantify the completeness of genomic data sets in terms of the expected gene content based on evolutionary principles. OrthoDB's sets of Benchmarking Universal Single-Copy Orthologs, BUSCO (http://busco.ezlab.org/) , provide a rich source of data to assess the quality and completeness of genome assemblies, gene annotations and transcriptomes. BUSCO quality assessment facilitates

informative comparisons, for example of newly sequenced draft genome assemblies to those of model species, or can be used to quantify iterative improvements to assemblies or annotations [4, 5]. BUSCO has rapidly become well-established as an essential genomics tool, using up-to-date data from many species present in OrthoDB, and with broader utilities than the once popular, but discontinued, Core Eukaryotic Genes Mapping Approach (CEGMA) [6].

2. Universal single copy genes

Protein coding genes that make up the BUSCO data sets are evolving under "single-copy control" [7], and are selected from OrthoDB [8] orthologous groups that contain genes present as single-copy orthologs in at least 90% of the species included in the group (Fig. 1). While allowing for rare gene duplications or losses, this establishes an evolutionarily informed expectation that these genes should be found as single-copy orthologs in any newly-sequenced genome or gene set from that group. Therefore, if there are many BUSCO genes from the appropriate clade that cannot be identified in a genome assembly or annotated gene set, it is possible that the sequencing and/or assembly and/or annotation approaches have failed to fully capture the complete expected gene content. Lineage assessments are available for vertebrates, arthropods, fungi, nematodes, plants, protists (see Subheading 3.2 and Table 1) and prokaryotes.

BUSCO uses sequence profiles embedded in lineage-specific datasets to assess the orthology status of predicted genes in the species under analysis. These consensus sequences are derived from Hidden Markov Model (HMM) profiles built from multiple sequence alignments of orthologs selected from OrthoDB, and capture the conserved alignable amino acids across the species set, reducing any potential species bias that would result from pairwise alignments towards original sequences. Current available lineages have been selected based on their taxonomic range and coverage in terms of the numbers of available sequenced and annotated genomes. As more and more species are sequenced and included in OrthoDB, we are going to update BUSCO assessment datasets and provide sets for new lineages.

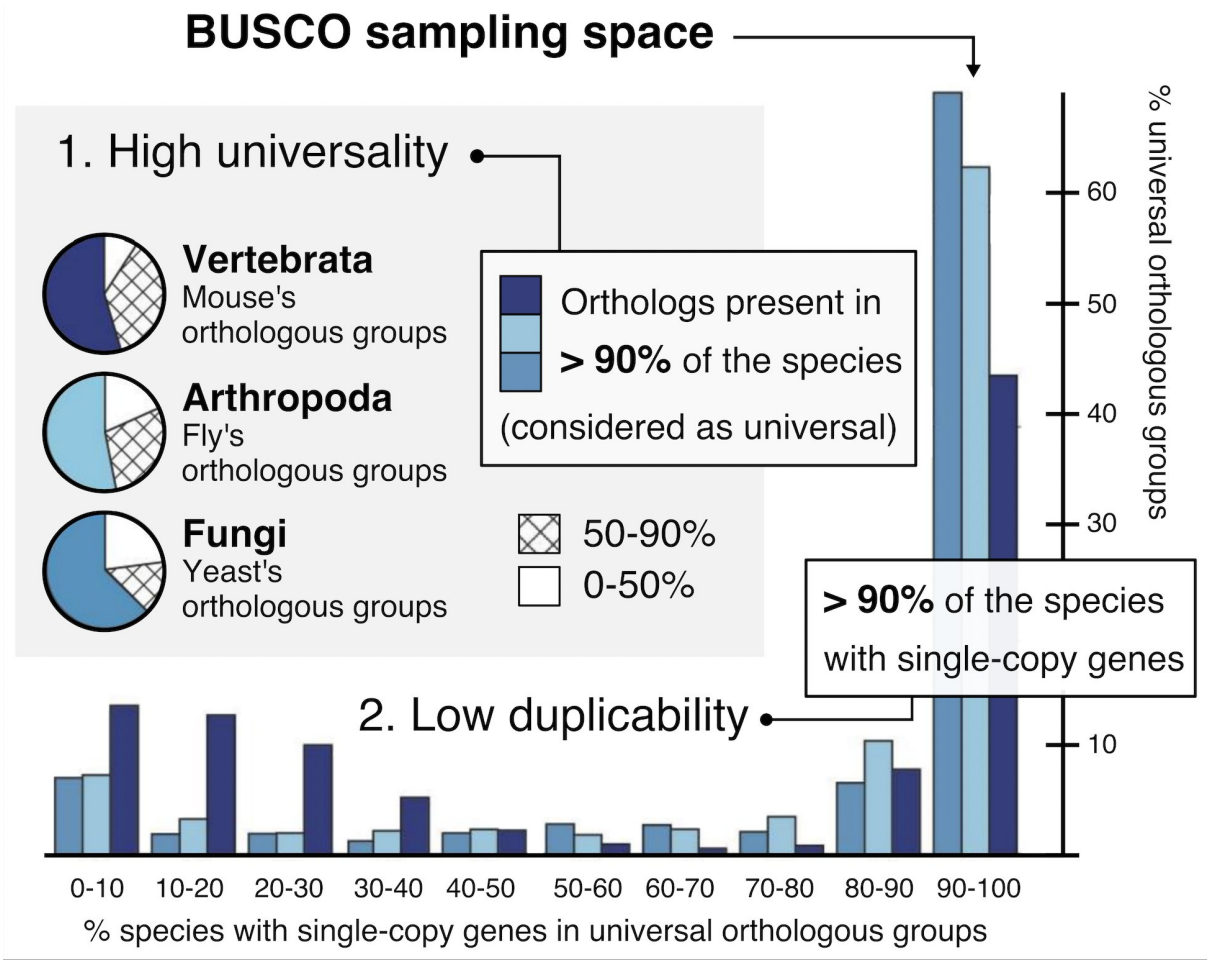

Fig. 1. BUSCO genes are selected among groups of orthologs matching specific evolutionary expectations. A BUSCO group has to encompass at least 90% of the species within its corresponding lineage, therefore showing a high universality, and be maintained as single-copy in at least 90% of the species to fulfill the low duplicability requirement. To illustrate this, the proportions of orthologous groups in mouse (Mus musculus), fly (Drosophila melanogaster), and yeast (Saccharomyces cerevisiae) are depicted according to the presence of orthologs in the other species within their respective lineage (pie charts: vertebrata, arthropoda, fungi) and the proportion of single-copy predominance in universal groups. The BUSCO sampling space is restricted to those passing both 90% thresholds. Adapted from [7].

3. The BUSCO software

BUSCO assessment procedure (Fig. 2) is implemented as a python 3 (https://www.python.org/) package built upon several third party tools, each performing one step of the global analysis to characterize BUSCO orthologs. The kind of input sequence (a genome assembly, an annotated gene set, or a transcriptome assembly) defines which of these steps need to be executed, namely *(i)* locate candidate regions using local alignment against amino acid consensus sequences, *(ii)* extract gene models from these regions based on block profiles, and *(iii)* score the candidate genes against the profile Hidden Markov Model (HMM) of their corresponding BUSCO genes. The tools are pipelined within three assessment modes offered to the user. The methods and examples presented hereafter are based on the version 3.0.x of the BUSCO open source software.

3.1. Setup

The BUSCO sources are hosted on https://gitlab.com/ezlab/busco/ where they can be downloaded or cloned using a *git* client. A mock input (*sample_data/target.fa*), an example lineage (*sample_data/example*), and the corresponding BUSCO genome evaluation results are available at the root of the repository and can be used to test and validate the installation of each component required to run a complete analysis.

3.1.1. BUSCO python package

The python package needs to be installed by calling the script *setup.py*. All python 3 versions are supported. In the main folder, one of the following commands has to be executed:

```
sudo python setup.py install     # system wide installation
python setup.py install --user   # current user only
```

The user is encouraged to pay attention to which version is used when calling *setup.py*, to ensure that the same version will be used when running the analysis.

3.1.2. Configuration file

BUSCO uses the *configparser* class from the python standard library to manage its configuration in a dedicated file organized in sections. The first section contains all parameters controlling the BUSCO run, while additional sections locate executables which are part of external tools. Below is an extract of the file *config.ini.default*, a self-documented default configuration present in the *config/* folder, which can be copied and adapted by the user for their own need.

```
[busco]
debug = True
gzip = False
[tblastn] # section describing the "tblastn" executable
path = /usr/bin/ # do not append the executable to the path
```

The path to the configuration file has to be declared in the *$BUSCO_CONFIG_FILE* environment variable or placed to the default location inside the BUSCO directory: *config/config.ini*.

3.1.3. Third party software

Each of the following tools has to be installed prior to running BUSCO in an assessment mode that requires it. The path to each executable has to be declared in the configuration file. It is recommended that the user makes sure that each of the software packages work independently before attempting to run any assessments with BUSCO. The minimal version that is required is mentioned for each tool, and the user can obtain information about future version compatibility on the BUSCO website and on the web pages of each tool.

A) TBLASTN

The genome and transcriptome modes require a translated BLAST search (TBLASTN) [9]. BUSCO uses the NCBI implementation available in the BLAST+ suite from version 2.2.x (see Note 1). It can be downloaded from https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/, and release notes are available on https://www.ncbi.nlm.nih.gov/books/NBK131777/. Two executables have to be declared in the configuration file: *makeblastdb* and *tblastn*.

B) AUGUSTUS gene predictor

AUGUSTUS, a tool for predicting genes in eukaryotic genomic sequences [10], is needed by the genome assessment mode. BUSCO supports versions 3.2.1 or higher and the software can be obtained from http://bioinf.uni-greifswald.de/augustus/. As it includes multiple PERL scripts (https://www.perl.org/), the user should refer to the AUGUSTUS documentation for its PERL requirements. The executables that have to be declared in the configuration file are *augustus, etraining, gff2gbSmallDNA.pl, new_species.pl,* and *optimize_augustus.pl*. These entries are not sufficient and additional environment variables have to be set as follows:

```
export PATH=/path/augustus-3.x.x/bin:$PATH
export PATH=/path/augustus-3.x.x/scripts:$PATH
export AUGUSTUS_CONFIG_PATH=/path/augustus-3.x.x/config/
```

(see Note 2)

AUGUSTUS makes its predictions based on parameters that are species-specific. It comes with predefined values corresponding to well annotated species. While BUSCO preselects automatically the most appropriate species to be used for each analysis, it is worth mentioning that these are listed in *$AUGUSTUS_CONFIG_PATH/species/* and it is possible for the user to indicate a different species (see Subheading 4.2).

C) HMMER

To evaluate amino acid sequences using profile HMMs, all modes of BUSCO require HMMER, version 3.1b2 or higher [11], which can be obtained on http://hmmer.org/. The unique executable that has to be declared in the configuration file is *hmmsearch*.

3.2. BUSCO Datasets

Analyses are based on features describing BUSCO genes that were carefully selected (Fig. 1). They are organized in datasets corresponding to specific lineages [5]. The version 3 of BUSCO comes with 28 eukaryotic datasets (Table 1) representing major groups, which can be downloaded on the BUSCO website along with 16 prokaryotic sets. They are identified by a name and a version, e.g.

“eukaryota”_“odb9”. Each BUSCO gene, with its unique identifier, e.g. EOG090C01CE (see Note 3), is represented by different parameters found in different files. The content of a standard BUSCO dataset is the following:

- **hmms/**: contains one profile HMM file for each BUSCO gene. Required by HMMER.
- **info/**: contains the list of species used to create the set and additional information.
- **prfl/**: contains one block profile file for each BUSCO gene. Required by AUGUSTUS.
- **ancestral**: a FASTA file for each BUSCO gene, which contains a consensus of the extant sequences. Required by TBLASTN.
- **ancestral_variants**: a FASTA file for each BUSCO gene, which contains a consensus and variants of the extant sequences. Required by TBLASTN.
- **dataset.cfg**: configuration and information about the dataset, including the default species used by AUGUSTUS among those provided with the tool, which corresponds to the most appropriate for the majority of species within the lineage, e.g. “fly” for the Insecta dataset.
- **scores_cutoff**: minimal HMMER scores to reach for each gene to be considered as orthologous to BUSCO genes and classified as found.
- **lengths_cutoff**: minimal length values for BUSCO gene matches to be called complete.

3.3. Running BUSCO

3.3.1. Genome

When the input to be evaluated is a genome assembly, i.e. nucleotide sequences in the forms of contigs, scaffolds, or chromosomes, the genome mode has to be selected (Fig. 2A). It runs two phases composed of three steps each. In the beginning of the first phase, TBLASTN is run taking BUSCO amino acid consensus sequences as queries and the input genomic sequences as database. The goal is to identify the subset of sequences in this genome that are most likely to contain matches for each BUSCO gene. Second, AUGUSTUS is run to delineate precise gene structures on these regions, from which a protein sequence is extracted. Finally, HMMER is run to assign a score to the candidate

amino acid sequence before the BUSCO algorithm proceeds to a preliminary classification. The second phase of BUSCO genome mode involves a retraining step, which produces a better set of parameters for AUGUSTUS, inferred from the single copy BUSCO genes found to be complete during the first phase. The rest of the run focuses on finding the missing BUSCO genes with a TBLASTN step based on additional variants of the amino acid consensus, followed by an AUGUSTUS step using the retrained parameters, and a new HMMER run to obtain a final classification.

The BUSCO genome mode is run as follows:

```
python busco_folder/scripts/run_BUSCO.py
-i SEQUENCE_FILE.fna -o OUTPUT_NAME
-l lineages/NAME_OF_LINEAGE -m geno
```

Every BUSCO mode displays a printed score on the stdout and produces a comprehensive output folder named run_*OUTPUT_NAME*. The following files and folders are found in a BUSCO genome run output:

- ***short_summary_OUTPUT_NAME.txt***: a text file that contains the final BUSCO score and a summary of the parameters that were used.

  ```
  # The lineage dataset is: NAME_OF_LINEAGE
  # BUSCO was run in mode: genome
  C:80.0%[S:80.0%,D:0.0%],F:0.0%,M:20.0%,n:10
  8 Complete BUSCOs (C)
  8 Complete and single-copy BUSCOs (S)
  0 Complete and duplicated BUSCOs (D)
  0 Fragmented BUSCOs (F)
  2 Missing BUSCOs (M)
  10 Total BUSCO groups searched
  ```

- ***full_table_OUTPUT_NAME.tsv***: The detailed list of all BUSCO genes and their predicted status in the genome.

  ```
  # Busco id      Status     Contig     Start End   Score Length
  EOG09000001     Complete   sample     3018  3142  320   193
  EOG09000002     Complete   sample     3164  4762  872   443
  ```

- ***missing_buscos_list_OUTPUT_NAME.tsv***: the list of missing BUSCO gene identifiers.
- **blast_output/**: contains the raw output of the two TBLASTN runs and the corresponding

coordinates as defined by BUSCO to represent candidate regions.

- **augustus_output/**: contains a log file dedicated to AUGUSTUS, the list of single copy genes that were used to retrain AUGUSTUS, and in the subfolder *predicted_genes/*, one gene model for each candidate region evaluated, named after the BUSCO block profile used, e.g. *EOG09000001.out.1*. In the subfolder *extracted_proteins/*, there are one nucleotide and one amino acid sequence for each gene model, e.g. *EOG09000001.fna.1* and *EOG09000001.faa.1*. Note that the two previously mentioned subfolders represent all candidates, including those that were not retained as positive matches in the end of the analysis, therefore containing irrelevant material that is not listed in the final full table file. Consequently, the user should be cautious when considering their content as meaningful biological sequences and refer to the coordinates and identifiers in the full table file. The retraining parameters produced by BUSCO to be used by the second AUGUSTUS run are stored in the subfolder *retraining_parameters/* (see Note 4). Finally, several intermediate files produced during the analysis in GenBank and General Feature Format (GFF) can be found in the remaining folders.
- **hmmer_output/**: contains a tabular format of each HMMER output, one for each candidate protein evaluated, named after the BUSCO profile HMM used, e.g. *EOG09000001.out.1*. These represent all candidates that were and were not retained as positive matches in the end of the analysis.
- **single_copy_busco_sequences/**: contains the nucleotide and amino acid sequences of all BUSCO genes that were found complete and not duplicated during the first phase of the BUSCO genome analysis. They are the genes used to train custom AUGUSTUS gene models. To access the genes that were found in the second phase, or found duplicated and fragmented during both phases, the user will have to manually extract the sequences from the other folders, according to the coordinates and identifiers in the full table file.

3.3.2. Annotated gene set

When the input to be evaluated is an annotated gene set in the form of amino acid sequences, the

protein mode has to be selected (Fig. 2B). It consists of a single assessment of every sequence against every BUSCO profile HMM followed by the classification. Annotated gene sets usually contain protein isoforms that are relevant and therefore kept in the final result. However, in order to properly evaluate the amount of BUSCO gene duplications (which can be technical artifacts or true duplications), isoforms should be removed before any BUSCO assessment.

The BUSCO protein mode is run as follows:

```
python busco_folder/scripts/run_BUSCO.py
-i SEQUENCE_FILE.faa -o OUTPUT_NAME
-l lineages/NAME_OF_LINEAGE -m prot
```

The BUSCO protein run output folder contains a *short_summary_OUTPUT_NAME.txt* and a *missing_buscos_list_OUTPUT_NAME.tsv* file identical to those of the genome mode. The *full_table_OUTPUT_NAME.txt* file is slightly different, having the identifier of the sequence and no start and end coordinates.

```
# Busco id      Status     Sequence   Score Length
EOG09000001     Complete   sample1    320   193
EOG09000002     Complete   sample2    872   443
```

The folder *hmmer_output/* contains a tabular format of each HMMER output, one for each BUSCO profile HMM that has been searched, e.g. *EOG09000001.out.1*.

3.3.3. Transcriptome

The last mode available has to be selected when the input is a transcriptome assembly (Fig. 2C) in the form of nucleotide sequences representing individual transcripts. A TBLASTN run taking BUSCO amino acid consensus sequences as queries and the input transcripts as database is conducted to obtain a subset of sequences harboring potential matches to each BUSCO gene. A six frame translation is done on these transcripts and HMMER is run to assign a score to the candidate amino acid sequences before the BUSCO algorithm proceeds to the final classification. As for protein isoforms, alternate transcripts should be removed from the input before running BUSCO in order to obtain a meaningful duplication score.

The BUSCO transcriptome mode is run as follows:

```
python busco_folder/scripts/run_BUSCO.py
-i SEQUENCE_FILE.fna -o OUTPUT_NAME
-l lineages/NAME_OF_LINEAGE -m tran
```

The BUSCO transcriptome run output folder contains a *short_summary_OUTPUT_NAME.txt* and a *missing_buscos_list_OUTPUT_NAME.tsv* file identical to those of the two previously described modes. The *full_table_OUTPUT_NAME.txt* file contains the identifier of the transcript and is similar to that of the protein mode. The folders *blast_output/* and *hmmer_output/* have the same content as their equivalents in the genome mode. The folder *translated_proteins/* contains the six-frame translated version of every transcript having a match to a BUSCO amino acid consensus during the TBLASTN analysis, including discarded candidates and transcripts included in the final classification.

3.3.4. Optional arguments

The BUSCO script possesses several options that allow the user to either act on the assessment outcome by fine tuning parameters, or control the usability of the tool, affecting the structure of the outputs or the resources and time consumption. While the first category will be evoked later in the chapter, it is worth highlighting useful options belonging to the second category. The full list of parameters can be printed by calling the help option:

```
python busco_folder/scripts/run_BUSCO.py -h
```

In addition to the command line, most of the parameters can be defined in the configuration file to become the default value at each run. For example:

```
python busco_folder/scripts/run_BUSCO.py --cpu 8
```

is equivalent to having in the configuration file:

```
[busco]
cpu = 8
```

Note that the parameters provided through the command line will always override the entry in the configuration file. A few parameters are restricted to the file, an important one being the *debug* mode that every BUSCO user should know.

```
[busco]
;debug = True # to enable, uncomment by removing the ;
```

Since BUSCO makes thousands of calls to external commands such as *augustus* or *tblastn,* it may be useful for the user to be able to track each of these calls and run them manually (see Note 5). Therefore, the *debug* mode prints all commands and parameters that are called during the run.

```
DEBUG ['/usr/bin/tblastn', '-db', '/tmp/test_db']
```

Once all issues related to external commands are fixed, or if the analysis was killed accidentally, the run can be started again from the beginning using the *--force* option to rewrite over existing files, or preserving the output of each step that was successfully completed using the *--restart* option. Finally, as each analysis generates a large amount of small files, some storage systems may be affected when multiple runs are conducted and kept during a project. The *--tarzip* option solves this issue by archiving all subfolders in the output that are likely to contain a high number of elements (i.e. AUGUSTUS and HMMER outputs).

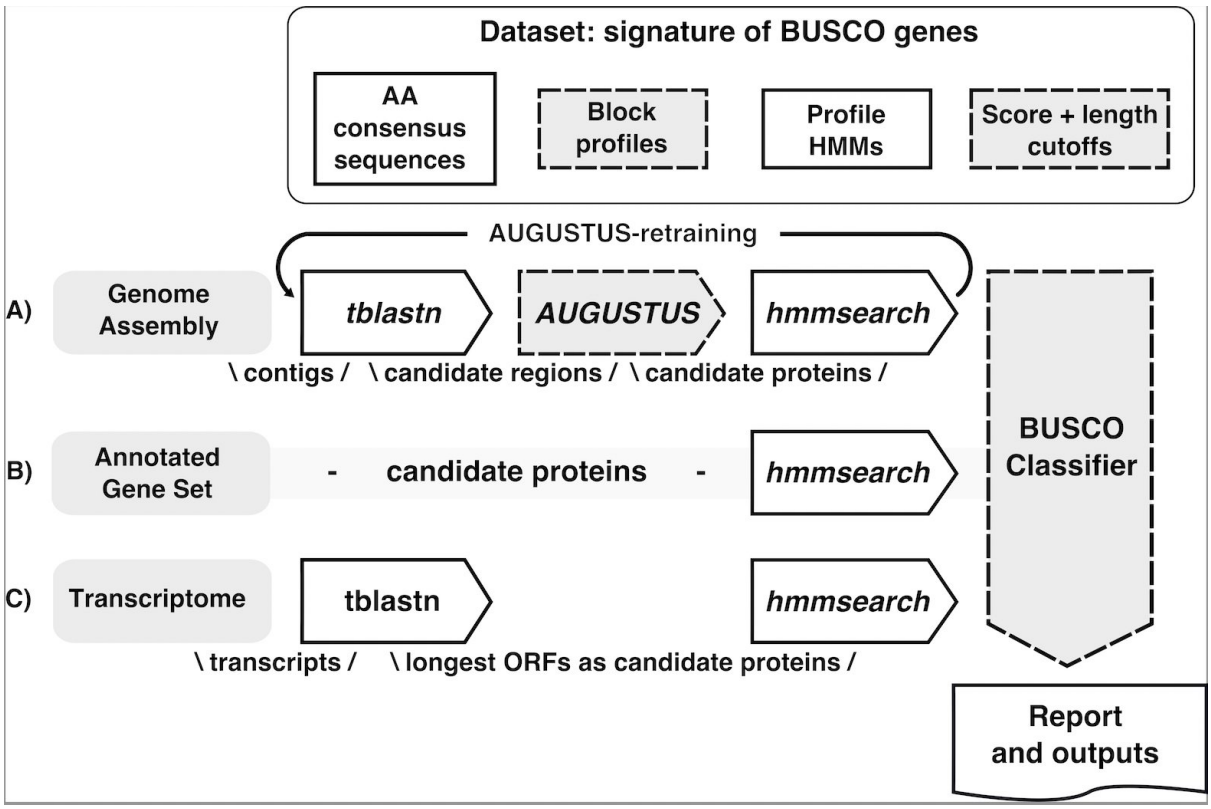

Fig. 2. Description of the BUSCO workflow for the three types of sequence input, genome (A), gene set (B) and transcriptome (C). The same dataset is used in all modes, although not all information

embedded is utilized in each situation. The genome mode includes two phases in which the three main steps are run, with the second pass only targeting the missing and fragmented BUSCO genes using additional consensus sequences and retrained AUGUSTUS parameters.

4. Understanding BUSCO

4.1. Choice of the dataset

Once the tool is properly set up, the first decision that has to be made is which dataset should be used among those available (Table 1, see Note 6). The primary goal of the BUSCO tool is to allow evaluation, comparison and reevaluation of assemblies and annotations. A good rule of thumb is to select the most specific lineage the species belongs to, as it will provide the best resolution possible for the evaluation. For instance, when working with insects, the user should choose the dataset belonging to the class Insecta, unless the organism belongs to the order Diptera or Hymenoptera for which an order-specific dataset exists [12]. While it is always incorrect to use any dataset issued from a lineage to which the species does not belong, two reasons could lead the user to select a more generic dataset, representing a higher taxonomic level. First, the time required by most steps during a BUSCO analysis increases linearly with the number of genes included in the dataset, which tends to increase in lower taxonomic levels. Moreover, species belonging to certain lineages such as Mammalia have a complex gene structure [13], which can drastically increase the run time per gene compared to a generic set such as Metazoa. Therefore, the user has to balance the resolution needed vs the runtime. Second, BUSCO is often used to compare an assembly or an annotation to previously published material of related species or a previous version of the same project. In this situation, it is recommended to plan in advance the comparative aspect of the project to select a dataset that encompasses all species involved. For example, while BUSCO provides an avian dataset, a comparative genomics study that mixes birds and other amniotes will prefer the Tetrapoda dataset for evaluating all species involved.

4.2. Choice of the parameters

BUSCO offers multiple ways to fine tune several aspects of the analysis. In particular, AUGUSTUS is a tool that provides many options and BUSCO can pass any parameter to this tool in the genome

mode using the option *--augustus_parameters*. However, the golden rule when using BUSCO is not to change default parameters unless there is a biological or experimental reason to do so. The user should keep in mind that their goal is not to improve the BUSCO score *per se*, but to improve the overall quality of their assembly and annotation, which also relies on remaining comparable with the rest of the BUSCO user community. One biological justification for editing a parameter is that of a different codon usage (the AUGUSTUS parameter *translation_table*), for example in ciliates [14], as keeping the default parameter would impair the evaluation of the assembled genome. However, if the goal is not to evaluate and compare, but to recover as many BUSCO gene sequences as possible for downstream analyses (see Subheading 6), it becomes relevant to explore the palette of parameters offered by BUSCO. The default species passed to AUGUSTUS can be specified with the *--species* option and the Expect value used with TBLASTN can be modified using the *--evalue* option. By default, BUSCO considers only the three best contigs matching a BUSCO gene during the TBLASTN step for the subsequent analyses to minimize the computing time. While this is an efficient tradeoff between performance and BUSCO gene recall for most use cases, the user can increase this limit up to 20 using the *--limit* option to try recovering a few extra BUSCO gene sequences.

4.3. Interpretation of the results

BUSCO produces a report for each of the three modes of assessment using the same scoring scheme. Expected BUSCO genes can fall into different categories: **C:complete** [**S:single-copy**, **D:duplicated**], **F:fragmented**, and **M:missing**. These are reported as absolute numbers as well as percentage of the total BUSCO genes (**n:**) included in the dataset. To judge whether a score is satisfying, the user will have to consider the type of sequence first. A very good genome assembly should contains all BUSCO genes that were not lost during the evolution of the species, which cannot be precisely defined. Model organisms, which have good reference genomes, often reach a score above 95% complete (BUSCO 3.0.2: *Mus musculus*; GRCm38.p6; mammalia_odb9; C:95.2%[S:90.9%,D:4.3%],F:2.4%,M:2.4%,n:4104 - *Drosophila melanogaster*; Drosophila melanogaster Release 6 plus ISO1 MT; diptera_odb9;

C:98.7%[S:98.2%,D:0.5%],F:0.8%,M:0.5%,n:2799 - *Saccharomyces cerevisiae*; Saccharomyces cerevisiae S288C; saccharomycetales_odb9; C:98.3%[S:97.7%,D:0.6%],F:0.7%,M:1.0%,n:1711). Non-model genome projects commonly report BUSCO scores ranging from 50% up to 95% complete, depending on the challenge posed by the species' biology (e.g. genome size, amount of repetitive elements) and its taxonomic position [15–18]. The score of an annotated gene set may reach a value lower than its genomic equivalent, since an annotation pipeline might miss BUSCO genes present in the assembly as it aims at predicting thousands of genes with broad parameters, while the BUSCO software targets very specific sequences with tailored parameters. Consequently, the user should assess both the assembly and the annotation result to judge whether the gene prediction strategy is appropriate or can be improved in light of the expected gene content in the genome (see Subheading 6.1). It is important to mention that the user should never complete the annotated gene set with the BUSCO genes recovered by the genome mode prior to assessing it, as it would bias the evaluation of true annotation efficiency. Finally, a good transcriptome score can be much lower than its genomic counterpart, as not all BUSCO genes are necessarily expressed together, especially in a single tissue or condition [19].

The duplication of a few BUSCO genes in a genome is compatible with a biological reality, as their evolution under single copy may be relaxed in some sublineages and the fact that we allowed duplications in up to 10% of the species when defining BUSCO markers [7]. However, a high duplication rate in a genome could denote a potential assembly of different haplotypes, a recent whole genome duplication [20], or technical artifacts that will have to be investigated. As mentioned earlier, the duplication rate of transcriptomes and annotated gene sets unfiltered for isoforms may be considerably higher. In some situation, the user will want to filter these out to decrease the duplication rate down to values expected in a genome. A high rate of fragmented BUSCO genes indicates issues in the sequencing and assembly process or the inability of the annotation pipeline to fully capture the complexity of some gene models. Turning fragmented BUSCO genes into complete is a good indicator of a significant improvement of the quality of an assembly, especially when supported by changes in other metrics such as N50.

To define the presence, absence and fragmentation status of each BUSCO gene, the classifier applies

to all results a score and a length threshold based on the distribution of these metrics in the species used to produce the datasets. This implies that in limited cases, it may be possible that a gene which is an outliers in terms of length or score will be classified as fragmented or missing while it is in fact present and complete. An advanced user may be able to spot such situations when manually investigating the outputs. However, much care should be taken when reinterpreting the results, as close homologs are sometimes difficult to distinguish from actual BUSCO genes (see Note 7) and remain the most likely explanation in such situations. Finally, no adjusted scores should be reported alone in a publication, for the sake of like-for-like comparisons within the community of users.

5. Plotting the results

It is common to represent BUSCO scores side-by-side using bar plots to illustrate different milestones of an assembly and annotation project, or different species as part of a comparative study. To encourage the use of a standard and distinctive layout in publications, while allowing a certain degree of customization, BUSCO includes a dedicated script to produce a figure and its source code that can be edited by the user. It requires only the short summary files of each BUSCO run that should appear on the plot to be grouped in a single folder, the working directory, in which the outputs will be generated.

```
python busco_folder/scripts/generate_plot.py -wd PATH
```

The language underlying the figure creation is R [21] and its popular library ggplot2 [22]. If these are available on the system running BUSCO, an image file will be produced automatically by calling the R script and written in the working directory. Otherwise, the user can specify the *--no_r* option to ignore this step and find a R code file in the working directory on which they have full control and freedom to edit and run anywhere. Fig. 3 is an example of the resulting default plot.

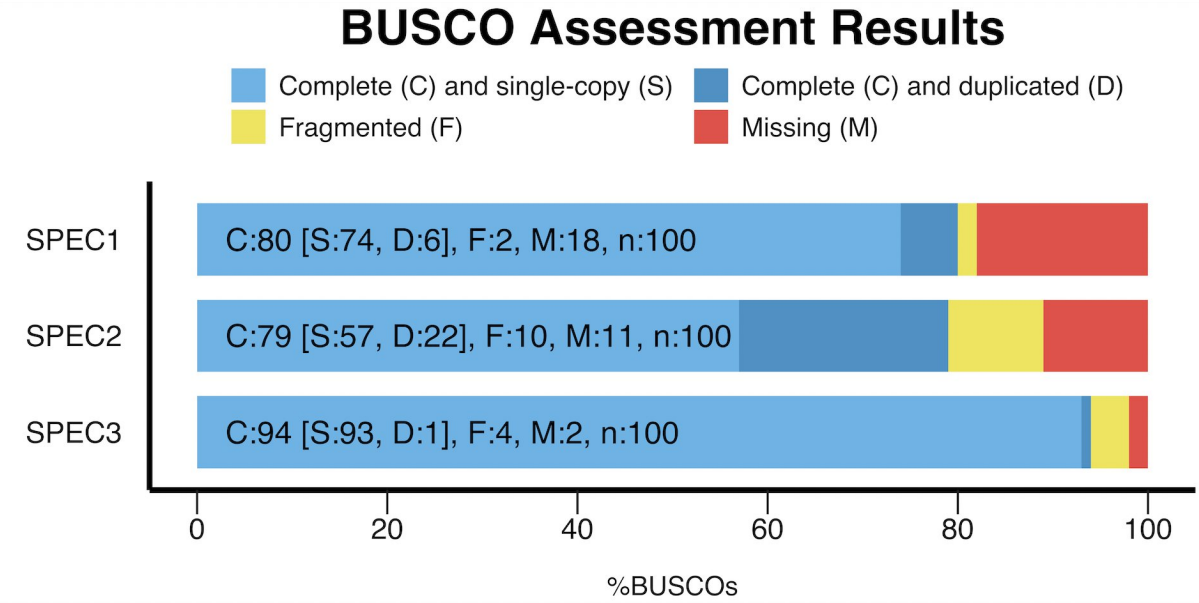

Fig. 3. Illustration of the BUSCO default side-by-side representation of assessment scores as produced by the plotting script. Three hypothetical species evaluated with 100 BUSCO gene profiles are depicted with various degrees of completeness and duplication.

6. Beyond completeness assessment

Although BUSCO's main function is to perform genomics data quality control, it is worth mentioning that one can take advantage of the pipeline for performing other common operations in genomics, such as for building training sets for gene predictors, identifying reliable markers for large-scale phylogenomics studies, and selecting high-quality reference species for comparative genomics analyses. These aspects are presented in great detail in the publication entitled "BUSCO Applications from Quality Assessments to Gene Prediction and Phylogenomics" [5].

6.1. A few words on gene predictor training

Running BUSCO provides to the user high-quality gene model training data that can greatly improve genome annotation procedures. Gene prediction remains a challenging procedure, especially in the absence of supporting evidence such as native transcripts or homologs from close species. To achieve the best results, gene prediction tools such as AUGUSTUS [10], SNAP [23], GENEID [24] and GeneMark [25], need to optimize their parameter configurations for each specific genome. BUSCO genes can be used as initial sets of high-quality gene models for such optimization. For example, using BUSCO-trained parameters for gene prediction resulted in improvements in the quality of the resulting gene model annotations over using available pre-trained parameters from other species [5]. Since BUSCO employs AUGUSTUS for gene prediction, the pipeline automatically provides AUGUSTUS-ready parameters trained on BUSCO genes identified as complete single copy (see Note 4). Moreover, BUSCO provides the *--long* option to enable the optimization mode when retraining AUGUSTUS, which can further improve the obtained retraining parameters, with a cost in terms of time consumption that depends on the complexity of the organism gene models. Other gene predictors like SNAP can be trained as well, by using as input the GFF and GenBank-formatted gene models generated by BUSCO.

7. Notes

1. Inconsistencies when using multi-threading on TBLASTN 2.4.x and higher have been reported multiple times. If the user faces such issue, a rollback to version 2.2.x or 2.3.x is a safe option. If this is not possible, BUSCO supports the option *blast_single_core=True* in the configuration file to ignore multithreading (*--cpu*) for blast steps only.
2. BUSCO needs to write in the $AUGUSTUS_CONFIG_PATH/species/ folder. Therefore, an unprivileged user on a shared environment will encounter the following error: Cannot write to AUGUSTUS config path. This can be solved by copying the entire $AUGUSTUS_CONFIG_PATH folder to a location where the user has write permission and redeclaring the environment variable to target this location.

   ```
   export AUGUSTUS_CONFIG_PATH=/new/location/
   ```

3. The BUSCO orthologous groups identifiers *EOGxxxxxxxx* cannot be shared or compared between different datasets and versions. The orthology delineation method uses a representation of the relationship between genes that is unique to each lineage as it arises from all duplication and speciation events underlying the evolution of the lineage. Therefore, a genomic sequence suitable to be a BUSCO gene in one dataset may not have the same orthology relationships to the sequences with different evolutionary distances that are considered to define BUSCO genes in other datasets.
4. To reuse the retraining parameters as a custom species with AUGUSTUS, independently from BUSCO, the user needs to move the folder *retraining_parameters/* back to the *$AUGUSTUS_CONFIG_PATH/species/* folder of their AUGUSTUS install and rename it to its original name, which can be deduced from its content. If the folder contains the file *BUSCO_OUTPUT_NAME_xxx_parameters.cfg*, the correct name to be used for naming the folder and identify the species within AUGUSTUS is *BUSCO_OUTPUT_NAME_xxx*.
5. BUSCO removes all temporary files at the end of the analysis. To run manually a command that accesses temporary files, the user will have to kill the run before it reaches the end.
6. Producing a BUSCO dataset is not a trivial task. Genes have to be sampled from orthologous

groups that are suitable in terms of phyletic profile (Fig. 1) and containing a sufficient number of species to properly represent the lineage in question. For this reason, and to encourage users to take advantage of existing datasets to produce comparable results, no detailed procedure for creating custom datasets is available. This remains achievable by an advanced user having access to a good sample of orthologs from their lineage of interest.

7. When close homologs to BUSCO genes are present in the sequence that is analyzed, the BUSCO classifier will give a better score to the true copies and therefore be able to discard the other sequences. However, if the actual BUSCO is missing, close homologs may sometimes reach a sufficient score to be considered as positive matches to a BUSCO gene that is in fact not present.

8. Acknowledgements

We would like to thanks all members of the Zdobnov group, in particular Felipe Simão and Christopher Rands for their useful comments. This work was partly supported by the Swiss Institute of Bioinformatics SER funding and the Swiss National Science Foundation funding 31003A_166483 to EZ.

Tables

Table 1. Detailed list of every BUSCO eukaryotic datasets available with BUSCO v3. The taxonomic ranks match the NCBI taxonomy browser classification (https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi).

Name	Taxonomic rank	Number of genes	Consensus of n OrthoDB species
eukaryota_odb9	Domain	303	65
metazoa_odb9	Kingdom	978	65
nematoda_odb9	Phylum	982	8
arthropoda_odb9	Phylum	1,066	60
insecta_odb9	Class	1,658	42
endopterygota_odb9		2,442	35
diptera_odb9	Order	2,799	25
hymenoptera_odb9	Order	4,415	25
vertebrata_odb9		2,586	65
actinopterygii_odb9	Superclass	4,584	20
tetrapoda_odb9		3,950	55
aves_odb9	Class	4,915	40
mammalia_odb9	Class	4,104	50
euarchontoglires_odb9	Superorder	6,192	25
laurasiatheria_odb9	Superorder	6,253	25
fungi_odb9	Kingdom	290	85
microsporidia_odb9	Phylum	518	14
dikarya_odb9	Subkingdom	1,312	75
ascomycota_odb9	Phylum	1,315	75
saccharomyceta_odb9		1,759	70
pezizomycotina_odb9	Subphylum	3,156	50
sordariomyceta_odb9		3,725	30
eurotiomycetes_odb9	Class	4,046	25
saccharomycetales_odb9	Order	1,711	30
basidiomycota_odb9	Phylum	1,335	25
embryophyta_odb9		1,440	20
alveolata_stramenophiles_ensembl		234	24
protists_ensembl		215	33

References

1. Vurture GW, Sedlazeck FJ, Nattestad M, et al (2017) GenomeScope: fast reference-free genome profiling from short reads. Bioinformatics 33:2202–2204 . doi: 10.1093/bioinformatics/btx153

2. Chikhi R, Medvedev P (2014) Informed and automated k-mer size selection for genome assembly. Bioinformatics 30:31–37 . doi: 10.1093/bioinformatics/btt310

3. Hunt M, Kikuchi T, Sanders M, et al (2013) REAPR: a universal tool for genome assembly evaluation. Genome Biol 14:R47 . doi: 10.1186/gb-2013-14-5-r47

4. Simão FA, Waterhouse RM, Ioannidis P, et al (2015) BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. Bioinformatics 31:3210–3212 . doi: 10.1093/bioinformatics/btv351

5. Waterhouse RM, Seppey M, Simão FA, et al (2018) BUSCO Applications from Quality Assessments to Gene Prediction and Phylogenomics. Mol Biol Evol 35:543–548 . doi: 10.1093/molbev/msx319

6. Parra G, Bradnam K, Korf I (2007) CEGMA: a pipeline to accurately annotate core genes in eukaryotic genomes. Bioinformatics 23:1061–1067 . doi: 10.1093/bioinformatics/btm071

7. Waterhouse RM, Zdobnov EM, Kriventseva EV (2011) Correlating traits of gene retention, sequence divergence, duplicability and essentiality in vertebrates, arthropods, and fungi. Genome Biol Evol 3:75–86 . doi: 10.1093/gbe/evq083

8. Zdobnov EM, Tegenfeldt F, Kuznetsov D, et al (2017) OrthoDB v9.1: cataloging evolutionary and functional annotations for animal, fungal, plant, archaeal, bacterial and viral orthologs. Nucleic Acids Res 45:D744–D749 . doi: 10.1093/nar/gkw1119

9. Camacho C, Coulouris G, Avagyan V, et al (2009) BLAST+: architecture and applications. BMC Bioinformatics 10:421 . doi: 10.1186/1471-2105-10-421

10. Keller O, Kollmar M, Stanke M, Waack S (2011) A novel hybrid gene prediction method employing protein multiple sequence alignments. Bioinforma Oxf Engl 27:757–763 . doi: 10.1093/bioinformatics/btr010

11. Eddy SR (2011) Accelerated Profile HMM Searches. PLOS Comput Biol 7:e1002195 . doi: 10.1371/journal.pcbi.1002195

12. Araujo NS, Santos PKF, Arias MC (2018) RNA-Seq reveals that mitochondrial genes and long non-coding RNAs may play important roles in the bivoltine generations of the non-social Neotropical bee *Tetrapedia diversipes*. Apidologie 49:3–12 . doi: 10.1007/s13592-017-0542-2

13. Keren H, Lev-Maor G, Ast G (2010) Alternative splicing and evolution: diversification, exon definition and function. Nat Rev Genet 11:345–355 . doi: 10.1038/nrg2776

14. Kollmar M, Mühlhausen S (2017) Nuclear codon reassignments in the genomics era and mechanisms behind their evolution. BioEssays 39:1600221 . doi: 10.1002/bies.201600221

15. Ioannidis P, Simao FA, Waterhouse RM, et al (2017) Genomic Features of the Damselfly Calopteryx splendens Representing a Sister Clade to Most Insect Orders. Genome Biol Evol 9:415–430 . doi: 10.1093/gbe/evx006

16. Holt C, Campbell M, Keays DA, et al (2018) Improved Genome Assembly and Annotation for the Rock Pigeon (*Columba livia*). G3 Genes Genomes Genet 8:1391–1398 . doi: 10.1534/g3.117.300443

17. Plomion C, Aury J-M, Amselem J, et al (2018) Oak genome reveals facets of long lifespan. Nat Plants. doi: 10.1038/s41477-018-0172-3

18. Armstrong EE, Prost S, Ertz D, et al (2018) Draft Genome Sequence and Annotation of the Lichen-Forming Fungus *Arthonia radiata*. Genome Announc 6:e00281-18 . doi: 10.1128/genomeA.00281-18

19. Carruthers M, Yurchenko AA, Augley JJ, et al (2018) De novo transcriptome assembly, annotation and comparison of four ecological and evolutionary model salmonid fish species. BMC Genomics 19:32 . doi: 10.1186/s12864-017-4379-x

20. Teh BT, Lim K, Yong CH, et al (2017) The draft genome of tropical fruit durian (*Durio zibethinus*). Nat Genet 49:1633–1641 . doi: 10.1038/ng.3972

21. R Core Team (2017) R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

22. Wickham H (2009) Ggplot2: elegant graphics for data analysis. Springer, New York

23. Korf I (2004) Gene finding in novel genomes. BMC Bioinformatics 5:59 . doi: 10.1186/1471-2105-5-59

24. Blanco E, Parra G, Guigó R (2007) Using geneid to Identify Genes. In: Baxevanis AD, Davison DB, Page RDM, et al (eds) Current Protocols in Bioinformatics. John Wiley & Sons, Inc., Hoboken, NJ, USA

25. Borodovsky M, Lomsadze A (2011) Eukaryotic Gene Prediction Using GeneMark.hmm-E and GeneMark-ES. Curr Protoc Bioinforma 35:4.6.1-4.6.10 . doi: 10.1002/0471250953.bi0406s35