Snakemake as a workflow engine for reciprocal BLAST analyses

Lukas Becker

A thesis presented for the degree of
Bachelor of Science

Supervisor: Prof. Dr. Gunnar Klau
Second Assessor: Dr. Nicolas Schmelling
Algorithmic Bioinformatics
Heinrich Heine University Düsseldorf
Germany
25th February, 2021

Acknowledgments

I would like to express my gratitude for Prof. Gunnar Klau whose encouragement and guidance throughout this study has been a great support. Furthermore, I owe special thanks to Dr. Nicolas Schmelling for his invaluable support, constructive feedback and continued assistance in my various questions. Thanks for the opportunity to work on this thesis and the experience to work on this cooperative topic. In addition, I would like to thank Philipp Spohr for his patience with my many questions and his advice and valuable guidance.

Thanks.

Abstract

In modern genomic analysis, homology describes the evolutionary relationship between two sequences. In addition to a diverse classification of homology, the key concept for evolutionary and comparative genomics is the dichotomous differentiation of homology into orthologous and paralogous sequences. Sets of orthologous genes are used to obtain information regarding gene functions and phylogenetic relationships because orthologs tend to retain their biological function. Various methods have been developed in order to identify orthologous genes. Typically, orthologous and paralogous gene relationships are disentangled by measuring and comparing sequence similarities within sequences of interest. More precisely, when searching orthologous sequences in different genomes, those sequences will most likely match each other as best hits when performing sequence similarity searches. The resulting sequences are called reciprocal best hits (RBHs). The identification of RBHs is the most common method to infer putative orthologs in comparative genomic studies. In order to use this method for finding orthologous sequences, various work steps have to be carried out, some of which can be solved straightforwardly by program code, without requiring considerable additional effort.

In this thesis we used the Basic Local Alignment Search Tool (BLAST) command-line program in combination with the workflow engine Snakemake and the web framework Django to develop an application that can be used for the inference of RBHs. Snakemake was used to develop a pipeline for the execution of the workflow steps, providing the core functionality of the application. Due to web based data deployment, pipeline execution and monitoring, the application enables an easy and user friendly customization of reciprocal BLAST analyses. The application and all necessary software and packages can get installed with Docker, which enables a utilization of the application on every system. As a local database option the non-redundant protein database from NCBI was utilized, however, additional databases can easily be integrated. In order to test the application, a reciprocal BLAST analysis of the circadian clock protein kaiA from *Synechococcus elongatus* PCC 7942 was conducted. The pipeline successfully identified 606 RBHs as putative orthologous sequences in a reliable time frame on a custom notebook computer. The intuitive usability of the developed application enables an easy-to-use execution of reciprocal BLAST analyses for scientists who are not familiar with programming.

Contents

1 Scientific background

1.1 Classification of homologous sequences

Modern genomics highly depends on comparative analyses, which are typically based on sequence similarity. Sequence similarity can give first clues about relation and function of newly sequenced genes (Pearson 2013). While investigating sequence similarities is already a technically difficult task, the precise delimitation and definition of this similarity represents an additional complex problem (Reeck et al. 1987; Koonin 2005). Biological sequences are described as homologous if they share more similarity than it would be expected by chance. As a result, the sequences are considered to share a common origin, an evolutionary ancestor, and did not arise independently. This excess of similarity is proven by statistical estimates, which are calculated differently depending on the algorithms used for evaluating the alignment scores (Pearson 1998). However, the absence of statistical significance within an alignment does not automatically imply that those sequences are not homologous (Fokkens et al. 2010). In those cases, profile based realignments as well as evolutionary intermediate sequences can help to elucidate homology.

1.2 The dichotomy of homology - paralogs and orthologs

Homologous genes share sequence similarities in their core domains. Functions for an unknown gene are usually first implied based on the highest similarity to known genes (Gerlt and Babbitt 2000). However, this inference is, while being the fastest method for assigning function to unknown genes, problematic as it is often based on a partial homology of one or more conserved domains, while neglecting e.g. other shorter domains of the unknown gene (Facchin et al. 2003).

This inaccuracy of the term homology requires more restrictive and exact definitions of sequence similarity in order to get a robust evolutionary classification. In general, the main events that drive gene evolution are vertical inheritance of gene sequences followed by speciation, gene duplication and gene loss, horizontal gene transfer (HGT), gene fusion, fission and rearrangements (Koonin 2005). While vertical inheritance describes the transmission of particular genes from parents to their offspring, HGT relates to the transfer of genetic material between two organisms that is not conducted via reproduction and vertical transmission of DNA. Thus, HGT allows arbitrary ancestors rather than a distinct last common ancestor of two genes. All these events build a complex web of relationships between genes and could explain similarities or dissimilarities. The most important relationships are described with the terms of paralogous and orthologous sequences (Koonin 2005; Tekaia 2016). Paralogs are genes related via gene duplication, while orthologous genes originated via a speciation event (Fitch 1970). This means that orthologous sequences arise from a single ancestral gene in the last common ancestor of the compared organisms, that undergoes some sort of speciation. This definition

can be problematic and restrictive when it comes to orthologous sequences within prokaryotes due to the occurrence of HGT. Nevertheless, the definition provides the most notable explanation for orthology, which will be used in this work. Most often orthologous sequences have similar functions, while paralogous sequences frequently exhibit a functional differentiation in the respective organisms (Zhang 2003; Altenhoff et al. 2012). Nevertheless, the simple dichotomy of homology into orthology and paralogy does not cover all logical implications of evolutionary scenarios that can lead to divergence or convergence of gene sequences. The classification of homology demands more precise terms and definitions (Sonnhammer and Koonin 2002).

In the following section I will outline the most important terms explaining gene relationships considering different evolutionary scenarios. Figure 1 shows a hypothetical phylogenetic tree of three species (A, B and C) with two ancestral genes (X and Y) and their descendants.

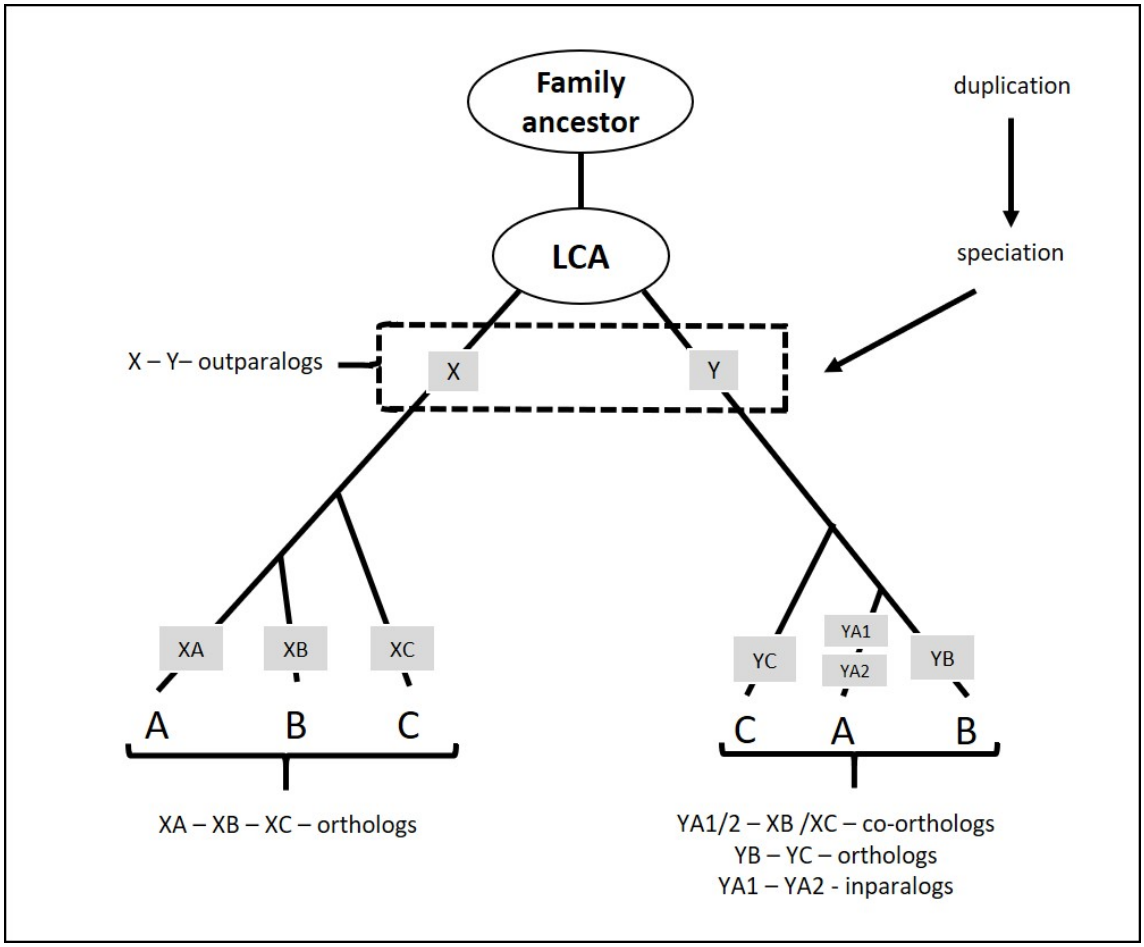

Figure 1: Hypothetical phylogenetic gene-tree of species A. B and C and genes X and Y. The tree describes the most common evolutionary events that lead to different gene relationships (modified after Koonin 2005).

The relationship of those genes outlines the potential differences within orthology and paralogy, which are mostly observed within scientific studies. The tree consists of two branches dedicated to the hypothetical genes X and Y. Each branch corresponds to a special case of

orthologous-paralogous relationship. Prior to the first speciation of the genes and the existence of the last common ancestor (LCA) a duplication event occured. This ancient duplication is the reason for the paralogous relationship of the two genes. Further speciation followed resulting in a radiation of the respective genes, thus, their gene relationship can get consolidated with the term outparalogs (Remm et al. 2001). Driven by vertical inheritance followed by speciation events the gene X diverged into three genes (XA, XB and XC), which are considered orthologous. This is the simplest case for inferring orthologs because there will be only one sequence in each species (A,B and C) that share a high level of sequence similarity between those genomes. In the branch of gene Y one can observe a speciation event followed by a lineage specific duplication in species A. The YA genes are co-orthologous to the genes of species B and C, while they are in-paralogous to each other. In addition, in cases of a gene duplication followed by a gene loss of one of the genes a reciprocal search can lead to a missclassification of a paralog as an ortholog. By examining the entire gene family of the given sequences it is possible to infer the correct relationship between the respective genes (Koonin 2005). The phenomenon of gene fission and fusion should also be considered when it comes to evaluating possible orthologs. Consequently it is possible that just specific parts or domains of a gene in one species are orthologous to different genes in other species (Song et al. 2008). Last but not least HGT can have an immense impact on inferring orthologs, especially when comparing prokaryotic genes due to its common occurrence in those organisms. If a species acquires a gene that displaces a gene, which has an orthologous relationship to another gene, the new gene could mimic an orthologous relationship to the other gene, because they do not fit into the definition of orthology as they do not share a last common ancestor. Those genes are referred to as xenologs (Koonin et al. 2001). This diverse classification of the term homology allows the correct assignment of the evolutionary relationship between two compared and analysed gene sequences.

1.3 Identification of orthologs and paralogs

The distinction between orthologs and paralogs allows a reliable functional annotation of newly sequenced genomes or sequences. It is a key aspect of comparative genomics, reconstruction of phylogenetic trees and in general, sequence analysis. Various methods have been developed in order to identify orthologous genes (Kristensen et al. 2011). The classical approach for identification of orthologs between different species is a phylogenetic analysis based on the comparison of gene trees and species trees. Once the two trees have been inferred they get reconciled or mapped on the basis of the parsimony principle, which results into a new parsimonious tree that should reflect gene relationships, hence orthologs and paralogs can be assigned by examining their relative position in the tree (Mirkin et al. 1995, Page and Charleston 1997a). Orthologs group more closely together with members from different species, while paralogs tend to group with members from the same species. In principle tree-based approaches of inferring orthologs are relatively robust and utilizable for disentangling

paralogs and orthologs. Furthermore, this phylogenetic method can take advantage of information stored in previously conducted multiple sequence alignments, which enables the inference of orthologs of an entire gene family. Two currently available applications for phylogeny based inference of orthologs are TreeFam (Li et al. 2006) and PhylomeDB (Huerta-Cepas et al. 2011). Both applications use slightly different phylogenetic approaches. TreeFam uses a curated database of phylogenetic trees for comparisons and statistical measurements. PhylomeDB takes advantage of a phylogenetic pipeline that involves statistical tests for choosing the optimal model for tree inference. In turn to those advantages there are also disadvantages that come with this technique (Page and Charleston 1997b). A major drawback is the occurrence of HGT, because gene tree topologies could differ dramatically from species tree topologies, this can directly lead to an evolutionary erroneous tree reconciliation and misinterpretation of gene relationships (Doolittle 1999). However, HGT is not determined as a major factor in eukaryotic evolution, besides there are more practical drawbacks of this phylogenetic approach. Applying this technique on whole genomes is computationally expensive and challenging due to the creation of trees for all genes and the execution of reliable statistical tests on those trees.

Given the drawbacks faced by those approaches, another method has been developed that is based on the simple assumption that orthologous genes are more similar to each other than to other genes. More precisely, when searching orthologous sequences in different genomes, those gene sequences will most likely match each other as best hits when performing sequence similarity searches (Tatusov et al. 1997; Remm et al. 2001). The resulting sequences are often called bidirectional, symmetrical or reciprocal best hits (RBH). The identification of RBHs is the most common method to infer putative orthologs in comparative genomic studies (Koonin 2005). RBHs can be determined by performing pairwise sequence similarity searches and evaluating the outputs by examining the ranking of homologous sequence sets. Fast, accurate and statistically confirmed algorithms for sequence similarity searches are needed to infer RBHs in comparative genomic studies.

1.4 Methods for identifying sequence similarity

There are different approaches for inferring sequence similarity, such as dynamic programming algorithms (DPAs), which assign scores to mismatches, matches, insertions and deletions in order to compute the least costly alignment between two sequences (Needleman and Wunsch 1970; Waterman 1984). The exhaustive use of DPAs are practical for comparing a small set of sequences but are impractical for similarity searching in large sequence databases for comparative genomic studies, given that aligning two sequences with those algorithms may be computed in time proportional to the product of their length (O(2)) (Pearson and Miller 1992). Based on these first algorithmic approaches, rapid heuristic algorithms have been developed for determining sequence similarity between genes. Such methods have enabled the possibility of sensitive similarity searches against larger sequence databases even on microcomputers due to the accomplished algorithmic (computational) efficiency (Waterman 1984).

1.4.1 BLAST is a fast and accurate heuristic algorithm for sequence similarity searches

Nowadays, the heuristic Basic Local Alignment Search Tool (BLAST) algorithm is a core application for inferring DNA and protein sequence similarity (Altschul et al. 1990). The algorithm can be used to identify homologous sequences between taxonomically distant species. It takes advantage of local sequence alignments to identify isolated regions of similarity in distantly related proteins. The algorithm has been developed in order to infer sequence similarity between newly sequenced genes and sequence databases. It computes similarity scores for two aligned segments of the same length, which are called “High Scoring Pairs” (HSP). The measurement of HSP scores involves statistical methods that use appropriate random sequence models that deliver a p-value. It predicts the possibility that two sequences randomly share similarities simply by chance (Karlin and Altschul 1990). The statistical measurement conducted by BLAST examines the distribution of alignment similarity scores within the underlying database. Thus, having more sequences in a database will increase the accuracy of the probability values. Explicitly, after a certain amount of comparisons, BLAST reports sequence alignments for the best scores. Those scores are assessed by a statistical value, the E-Value, which describes the expected number of times a similarity score would occur by chance. The BLAST algorithm incorporates a Mealy machine, a deterministic finite state model or finite state machine, while scanning the sequences of interests for high scoring pairs (HSP) (Altschul et al. 1990). The machine defines output values, e.g. scores of potential hits (HSP), based on the current state and input rather than solely on the current state. It is used to signal HSP acceptance on transitions, saving disc space and computation time nearly proportional to the size of the underlying alphabet (e.g. the protein or nucleotide alphabet). Once all HSP above or equal to a certain threshold are found BLAST tries to extend those hits into both directions. Each new extension is impacting the score by either increasing or decreasing it until the extension phase is reaching the end of the underlying sequences or if the score is below the determined threshold. If the alignment is above or equal to the predefined threshold it will be included into the alignment

table for the query sequence. Those technical, statistical and programmatic approaches make BLAST to a fast and reliable heuristic method to identify homologous sequences between taxonomically distant species (Altschul et al. 1990; Altschul et al. 1997).

Currently, BLAST is available on a web server (https://blast.ncbi.nlm.nih.gov/Blast.cgi) hosted by the National Center for Biotechnology Information (NCBI) or as a standalone command-line tool. Both versions come with numerous variations of the BLAST algorithm (s. Table 1) that provide adjusted strategies for all sets of biological sequences and search scenarios e.g. blastn and blastp, which are adapted for DNA and protein sequences. The current command-line BLAST applications are available to the public since 1997 (Altschul et al. 1997) and have been developed further in order to enhance computing time and functionality (Camacho et al. 2009). BLAST is one of the most heavily used sequence analysis tools and many different research teams are using the algorithm for inferring homologous sequences.

1.5 Reciprocal BLAST for inferring orthologous sequences

BLAST is an ideal algorithm for inferring RBHs and for detecting orthologs. RBHs can be assigned by performing a reciprocal BLAST. In a reciprocal BLAST sequence(s) of interest (SOI) from specific species for which orthologs should be found are used as input queries for the first sequence similarity search, the forward BLAST.

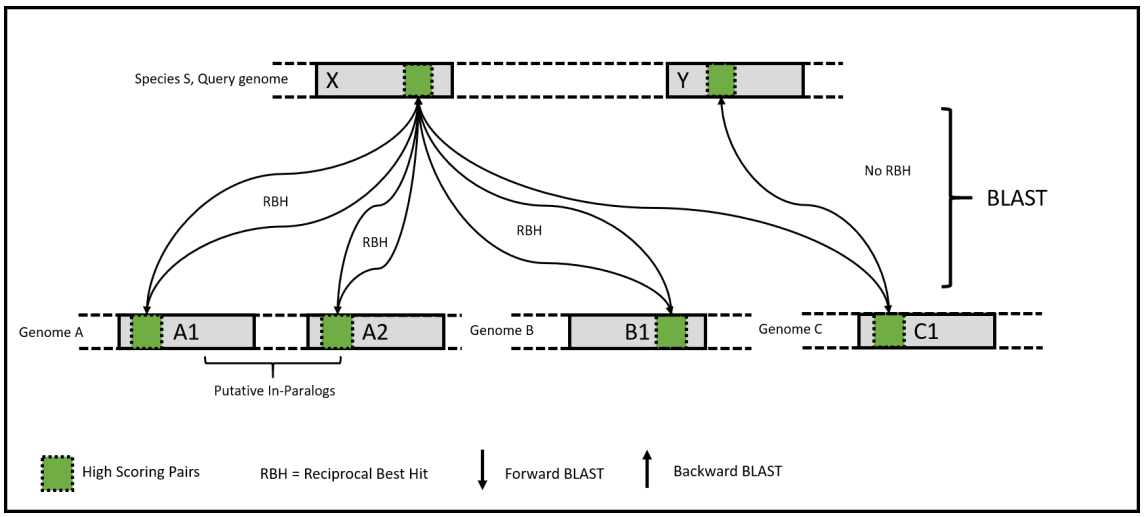

Figure 2: Example inference of RBHs during a reciprocal BLAST analysis. The example shows an inference of three putative orthologous sequences (A1, A2 and B1) and an homologous but not orthologous sequence (C1).

The generated output is analyzed and all sequence hits of the forward BLAST are extracted from the underlying database. Those extracted sequences serve as queries for the second sequence similarity search, the backward BLAST. This BLAST is limited by searching explicitly against all sequences from the species from which the forward BLAST queries come from. In addition, the output is limited to just report the best hit, or particularly the best alignment. Putative orthologous sequences are found if query and subject sequences from the forward

BLAST match against each other in the backward BLAST. For example, one would like to identify orthologous sequences from gene X of species S in genome A, B and C (Figure 2). In the forward BLAST a sequence similarity search is performed with gene X against all genes present in genome A, B, and C. The output of this forward BLAST may be the homologous sequences A1, A2, B1, and C1 that exhibit statistically significant similarities to gene X. Those sequences will now serve as input queries for the backward BLAST in which the genes A1 to C1 are compared to the genes present in the genome of species S. The output of this second BLAST is limited to just reporting the best matches. After the backward BLAST, a RBH can be assigned if, for example, gene A1 hits gene X as best hit. Contrary, if the gene C1 does not match with gene X (as best hit) but with another gene of species S such as gene Y there would be no reciprocal best hit for gene C1.

However, it has been revealed that the genes identified by BLAST best hits are not always each other's closest relatives in a phylogenetic point of view (Koski and Golding 2001). This finding led to the assumption that the BLAST score is not a good indicator to assign genes as orthologs in a simple one-way BLAST analysis. On the contrary, the reciprocal BLAST technique and the assignment of RBHs allows reliable identifications of orthologs. Furthermore, the utilization of local sequence alignments such as in BLAST can cope with the problem of gene fusion or fission (Karamichali et al. 2014; Henry et al. 2016). Genes that underwent such scenarios can share parts or domains that are orthologous, which are identified as HSP. In addition, it has been shown that even in comparing genomes from evolutionary distant species numerous genes form RBHs (Tatusov et al. 1996). Besides these advantages the RBH approach inherits also some disadvantages. Lineage specific duplication events are producing paralogous sequences. Those sequences can disturb the assignment of orthologs during a reciprocal BLAST, thus, putative orthologs can form best hits with the paralogous counterpart during the backward BLAST and the orthologous sequence will not be detected. This results in a false negative error because a pair of orthologous sequences is missed (Koonin 2005). The definition of orthologous sequences declares two sequences as orthologs if they share a common ancestor and RBHs can likely occur between two genes that share an excess of sequence similarity due to HGT events. Nevertheless, HGT does not influence the whole reciprocal BLAST analysis but only the false positive assignment of xenologs as orthologs. Most often xenologos are replacing former orthologs, furthermore, they adapt the functionality of the replaced ortholog and accordingly, xenology is most often a good predictor for functional equivalency, albeit xenologous genes do not fit into the definition of orthology.

1.6 Practical challenges in performing reciprocal BLAST analyses

For just a few sequences and quick analyses the currently most used method to infer homologs is the web hosted version of the NCBI BLAST tool (McGinnis and Madden 2004). This is most commonly used by wet-lab biologists without special training in bioinformatics or programming. For more sequences NCBI provides a command-line based tool that can run on local machines (Camacho et al. 2009). Besides remote search possibilities this tool provides the possibility to quickly search previously downloaded and study-specific databases for matches to query sequences. This command-line BLAST tool provides a variety of options that can be overwhelming and hard to use for researchers not familiar with programming and command-line software. However, for a reciprocal BLAST a second BLAST (the backward BLAST) and several other workflow steps are needed. This is an ideal situation for developing a pipeline that can execute different programs in a pre-designed sequence of tasks in order to avoid the error-prone manual execution of programs. Particularly, a reciprocal BLAST pipeline needs following workflow steps:

1. Process user defined input and configure settings for the reciprocal BLAST.
2. Execute the forward BLAST.
3. Examine output files and prepare input files for the backward BLAST based on all homologous sequences found by the forward BLAST.
4. Execute the backward BLAST.
5. Compare forward and backward BLAST and filter for reciprocal best hits.

These five steps are the core tasks for a reciprocal BLAST pipeline. The first step highly depends on the software that is used for executing the BLAST algorithm. Correct deployment of necessary input data such as the sequences for which orthologs should be assigned and their corresponding genomes is crucial for an accurate execution of the pipeline. The second and fourth step involve database preparation. This can be done through the command-line tool or without an appropriate database formatting tool, via a remote execution of BLAST on the NCBI BLAST web interface, that uses preformatted databases on NCBI servers. In addition, the third and fourth steps involve programming techniques in order to extract sequences out of genomes and to compare large data tables for the inference of RBHs. Once RBHs are assigned the results can get further processed in order to produce statistical reports and other project-specific information, e.g. the number of species for which orthologous sequences have been found or the amount of reciprocal best hits for each query sequence. Within the first step, data uploads are required for providing underlying genome databases for the BLAST searches. The remote procedure does not have this requirement as databases are stored on NCBI servers. For just a few query sequences executing remote BLAST searches on NCBI servers is an option but most of the time the pipeline should operate with numerous sequences and

whole genomes. On September 8, 2020, NCBI introduced new limits to web based BLAST analyses, the maximum number of target sequences cannot exceed more than 5.000 and the maximum allowed query length for nucleotide sequences is 1.000.000 and 100.000 for protein sequences (https://ncbiinsights.ncbi.nlm.nih.gov/2020/06/18/new-blast-settings/). In addition, servers of the NCBI BLAST web tool have a CPU usage limit for submitted searches. If the search exceeds more than one hour of combined CPU time the backend machines will stop the process. Hence, there is demand for providing biological databases on the machine on which the pipeline should be executed. There are a lot of sequence databases available with different levels of completeness. In order to allow an accurate reproduction of previous conducted pipeline executions, databases should only change on the explicit request of the user. These findings support a dynamic database resourcing, one that is conducted via the pipeline, at least for small databases, and another that involves a database resourcing before pipeline execution.

1.7 Snakemake as a workflow engine for reciprocal BLAST analyses

With currently available techniques the scientist would have to execute a set of different programs and specify numerous options for running the BLAST analyses, which is a time consuming, error-prone barrier especially for wet-lab biologists who are not familiar with command-line applications and programming. This situation is ideal for the integration of Snakemake. Snakemake is a workflow engine similar to the build-management tool GNU *make*, that can automate the execution of workflow operations (Köster and Rahmann 2012). It uses a Python based workflow definition language, which is close to pseudocode and easy to read and alter. Steps of the pipeline can be conducted via rules that are defined in a special text file, the snakefile. Due to the pseudocode-like syntax of those rules they can be easily altered. Furthermore, Snakemake provides the possibility to use configuration files that can get integrated into the execution of different snakefiles. This eases the transition of user input into the specified workflow executed by Snakemake. With default settings, Snakemake tries to execute the first rule in the snakefile. Further workflow execution is based on the access of input files, thus, a rule will not be executed if an input file is an output file of another rule. The rule execution will wait until all necessary input files are generated and finishes if all output files are generated. Thus, it is possible to define rules as build-targets, which serve as mounting points for rule execution. Snakemake looks for rules that produce the necessary input files and most often those rules depend on other input files that are produced by other rules, which will then get addressed by Snakemake. This procedure goes on until Snakemake finds a rule that can get executed. Owing to Snakemakes command-line options it is possible to create specific project directories such that for each project specific Snakemake configurations can be created. Once a project has been executed it would be easy to reproduce the generated data by re-executing the Snakemake workflow together with the specified project configuration. This enables the comparison of reciprocal BLAST analyses with different databases or

query sequences. In general, rules can easily be added and with that project specifications and requirements can get addressed. In the long term included rules can extend possible result processing procedures, e.g. construction of phylogenetic trees or annotation and functional classification of newly sequenced genes. In general, the most important rule definitions are rule names, input and output file specifications and a *shell* or *run* command. With Snakemakes *shell* parameter it is possible to use all local and global variables, thus, system variables can be used and external programs can get executed. Furthermore, it is possible to integrate external Python scripts into rules, which enables the full utility of this programming language. Python code must not reside in external scripts. By using the *run* parameter Snakemake allows the direct code execution within rules. Inside Python based rules it is also possible to invoke shell commands. This is done via the *shell()* function that takes a command string as argument. In combination with Python based rules this allows the chained execution of shell commands and Python functions. Snakemake offers additional rule parameters that can greatly enhance or enable more specific tasks. For example, the *notebook* parameter allows integration of Jupyter Notebook scripts, the *thread* parameter defines the number of threads usable by the rule, the *priority* parameter can be used to define rule specific scheduler priorities. Those parameters are just a few examples of options that can be used.

1.8 Research goals

The current available methods for inferring reciprocal BLAST analyses are based on remote execution possibilities via the NCBI web interface or on the command-line application. For both possibilities additional custom coding procedures need to be established in order for processing output files and generating input files. This procedure is error prone and complex due to the manual execution of functions or utilization of the web based BLAST interface from NCBI. Most wet-lab biologists are not familiar with coding and are not able to perform such analyses on their own. Furthermore, on September 8, 2020, NCBI introduced limits to their web based BLAST searches which complicate reciprocal BLAST analyses on whole genomes (s. Section 1.6). Thus, there is the need of an application that eases the execution of reciprocal BLAST analyses and enables whole genome comparisons. The aim of this project is to develop such a software. The application has to tie all necessary workflow steps together. Furthermore, it should enable an easy data deployment, monitoring options, reproducibility and in general a good user experience. Last but not least the software should be available for every operating system.

2 Material and methods

2.1 Programming language and packages

The project associated application is written in Python (version 3.9.0) (Van Rossum and Drake 2009). It uses different Python packages to handle the tasks that come with evaluating the input received from the user, storing this data into databases, executing the pipeline and presenting the results. These Python packages have been installed with the distribution platform anaconda (*Anaconda Software Distribution* 2020). Especially, BioPython (Cock et al. 2009) is used for evaluating the presence of scientific clades in the non-redundant (nr) database and converting scientific names into taxonomic nodes that can be used in combination with the *get_species_taxid.sh* script to limit BLAST sequence similarity searches by taxonomy. An overview of the used Python packages and versions can be found in the biosnakedjango-{system}.yml files, available on the git-repository of this project.

2.2 BLAST C++ command-line tool and NCBI data

The analysis for sequence similarity search for inferring homologous sequences is conducted with the BLAST command-line application (Camacho et al. 2009). The current available command-line tool offers diverse programs that can be used to maintain and update databases and to conduct different types of BLAST analyses. Following programs have been used within the developed pipeline: *blastp* and *blastn* for BLAST sequence similarity searches with possible changes to default settings, *makeblastdb* for BLAST database creation and *blastdbcmd* for retrieving sequences out of BLAST databases. Following settings of blastp or blastn can get altered by the user: *-evalue, -taxidlist, -word_size, -num_alignments, -num_descriptions, -num_threads* and *-max_hsps*. More information concerning the BLAST C++ application can be found on the BLAST user manual (Camacho et al. 2019). The executables used in this project can be downloaded from "ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.11.0/". Follow the installation instructions from the user manual. Additionally, the *get_species_taxids.sh* script from the BLAST command-line tool suite is used for limiting BLAST searches with high-level taxonomic nodes. BLAST only accepts taxonomic nodes that are below or at species level. The script translates higher level taxonomic nodes into a list of nodes that are at the appropriate level for the BLAST programs. The script depends on the Entrez Direct (E-Direct) toolset, which only works on Linux or Mac operating systems (Kans 2020). In addition to format databases during pipeline execution, the pipeline can get executed on preformatted databases. As a first option, the pre-formatted, compressed nr database was downloaded from the NCBI FTP server and decompressed on a local hard drive (ftp://ftp.ncbi.nlm.nih.gov/blast/db/). It is a protein sequence database for BLAST searches curated by NCBI. The database contains non-identical sequences from various other databases.

2.3 Snakemake as the core utility for pipeline execution

Snakemake is a workflow engine that can be used for enabling a chained execution of different command-line applications or programs (Köster and Rahmann 2012). This chained execution is controlled by rules that are defined in the snakefile. Before rule execution, rules are grouped into different jobs that depend on input and output files. Job grouping allows the establishment of an internal execution order by building a directed acyclic graph (DAG) and the possibility of a simultaneous execution of rules that do not depend on each other. Thus, jobs are created by Snakemake after evaluating the rule execution sequence. In this project Snakemake was used to execute a reciprocal BLAST pipeline, providing the core functionality of the developed application. The reciprocal BLAST analyses can be conducted against uploaded genome databases or against the preformatted nr protein database. A project-specific static snakefile was developed for each of the two project types. With the script parameter, some rules are executing external Python files that do reside in the same static directory. Both snakefiles use configuration files that are stored in the project-specific directories, which are established during project creation as requested by the user. Furthermore, these project directories serve as execution environments for Snakemake, hence all output files generated by Snakemake are written into that directory. Snakemake log files are used in the application for monitoring the pipeline process.

2.4 Django as web framework for application development

Django is an open source web framework written in Python that follows a model-view-template (MVT framework) scheme, which allows a rapid and robust implementation of web applications (*Django* 2019). It was used for the development of an user interface for project creation, data deployment, pipeline execution and monitoring.

Django models allow an easy class-based modeling of database tables. Models are written in the *models.py* file. Generally each model class maps to one appropriate database table. The underlying database is managed by Django. For this project a SQLite3 (Hipp 2020) database for the local development and a PostgreSQL (PostgreSQL 2021) database for the development with docker containers is used. SQLite3 and PostgreSQL are relational database management systems that follow SQL (Structured-Query-Language) database engine specifications. The Django models are directly linked to the data that is needed to execute the pipeline. In this work, the core model is the *BlastProject* class. Instances of this class are used as foreign keys for all other necessary models, e.g. *ForwardBlastSettings*, *BackwardBlastSettings* or *Genomes*, that are linked to the project. With an instance of the *BlastProject* class it is possible to get all relevant information for conducting the reciprocal BLAST analysis.

Views manage the HTTP methods received by the server from user requests. Views are written in the *views.py* file, each view function has its own distinct responsibility such as the project details view, which receives an user request and project_id and returns the current project-specific detail template. In turn, this detail template displays all of the project-specific data, such as the query sequence file and database names and the blast settings. Furthermore, if the pipeline execution finishes this view will present the resulting output data. Django will choose and execute a view function based on examining the URL that is requested, particularly on examining the part of the URL that comes after the domain name. Active URLs are stored within the *urls.py* files of your current project, hence all of the URLs can be found in the *blast_project/urls.py* file. Views can get rendered with instances of form classes. Fields inside those form classes can get validated and validation errors can get displayed to the user by views.

Django allows the utilization of templates, which provide a designer-friendly syntax for rendering the information that is presented to the user. They can generate any text-based formats such as HTML, XML or CSV. In this project the Django template language is integrated into HTML files. Templates can contain variables, tags and filters. Variables are replaced with corresponding values that are passed to the template inside the view functions. Similarly to common programming constructs, tags can control the logic of templates. For example, an “if” tag is used for boolean tests and a “for” tag for iterating over variables. Additionally, filters can be applied to variables which modify variables for display. For example, the “lowercase” filter converts text to lowercase.

2.5 Docker for application deployment

Different software packages and applications are necessary in order to perform the pipeline and to start the Django application. Parameters and functions that are executed can change depending on the software version. As a consequence, it is possible that an application will no longer work if different versions of required packages or software are used. Docker is an ideal solution for deploying software because it can containerize applications and manage the installation processes needed to deliver the desired functionality (Merkel 2014). Docker uses images for storing information utilized by containers, such that a container is an active instance of an image. Images can be built automatically by reading instructions from a text file called Dockerfile. Software and package versions inside an image cannot change once it is assembled. This enables the commitment on suitable software or package versions that can reside in an image.

Two images are used in this project, a PostgreSQL (PostgreSQL 2021) image for the database-server and an ubuntu:focal image for installing all fundamental software packages. Docker-compose is then used to tie both images together and to create a container-network. Uploaded data files are stored in volumes that are shared between the host machine and the web container. In the volume section of the *docker-compose.yml* file the relevant directories are listed. With default settings the application and project directories, the database directory, as well as an temporary directory for saving temporary files during upload processes of large genome files are listed as shared volumes. It is possible to add more directories if additional volumes should be shared between host and container, for example other database directories. In order to use the BLAST databases from the command-line it is important to specify the BLASTDB path variable within the container. This variable should point to one of the added volumes. With default settings the variable points to the "/blast/nr_database" directory within the container.

2.6 Guidelines

The source code for the reciprocal BLAST pipeline web application described in this thesis is accessible at: https://git.hhu.de/synmibi/reciprocal_blast_pipeline. This link was last checked on February 25, 2021. In this thesis we will be referencing version https://doi.org/10.5281/zenodo.4561364. The application is under development and the latest version can be found at: https://github.com/CyanoWorld/reciprocal_blast_pipeline.

2.6.1 Installation instructions

In order to start developing and using this application you need to download the application from the git-repository. There are two possibilities for installing the application either you install the package distribution platform anaconda or the container virtualization platform docker. The recommended installation procedure is the Docker setup. Especially windows users should install the application with docker, thus, automated deletion of project files will not work correctly on windows platforms and installation of the NCBI E-Direct tool is only possible on Mac or Linux operating systems. In order to enable the use of the pre-formatted nr database you have to download and decompress the database files from the NCBI FTP server (ftp.ncbi.nlm.nih.gov/blast/db/). A detailed installation routine is present at the git-repository of this project.

2.6.2 Developer Guidelines

Docker volume sharing between the host and container application directory enables development of this application inside the container. During container start the application gets started in development mode. In development mode Django uses a state reloader, which automatically reloads the development server whenever application files are changed. You do not need to stop the server and start it again if you make any code changes. In addition to Django's template syntax, HTML files are rendered with Bootstrap 4, which is an HTML, CSS and JavaScript framework primarily for designing websites. A good overview of all design elements and options Bootstrap offers can be found on the project page (*Bootstrap 4* 2020). A more detailed developer guide is present at the git-repository of this project.

2.6.3 User Guidelines

The user can access the application from the browser by typing the ip-address of the host server in combination with the port on which the application was started and the name of the URL. With default settings the port number is 8080, thus, the user can access the login page by typing ip-address:8080/blast_project into the search bar of any browser. The user needs to be registered in order to enable the login. After successful login the user is able to reach all sites of the application.

3 Web tool for reciprocal BLAST analyses

3.1 Project and source code directory structure

Due to the utilization of Django as a web framework the project design is clean and pragmatic. Thus, necessary settings such as the database and other core utilities are separated from application specific files such as the files for Snakemake execution or front end management. Furthermore, files that are generated by the application reside in their own distinct project directories within the media directory. The static directory contains two sub-directories for the snakefiles and the corresponding Python scripts that are executed by snakemake.

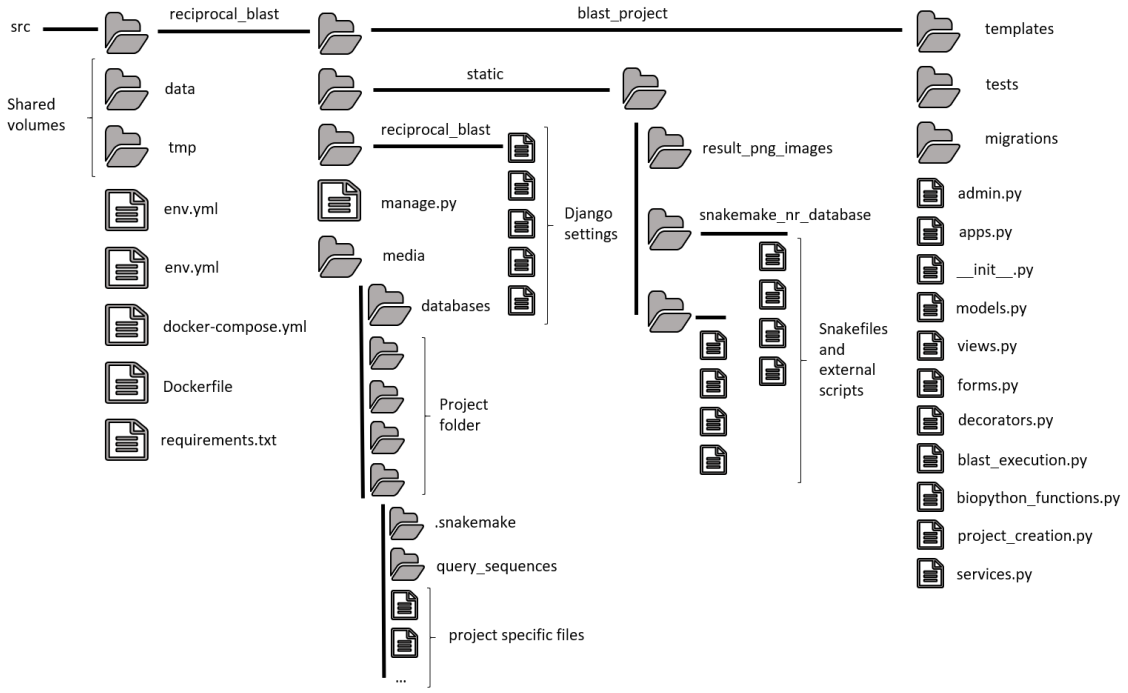

Figure 3: Overview of source code and project directories. The media directory contains all data that is generated by the application. Python files for application settings, as well as Snakemake specific files are separated as higher-level directories.

Python files for view, form, model and service code, which provide the core functionality of the application reside inside the blast_project directory. Some of the code files are service files for functions, that are imported and utilized in the *views.py* and *forms.py* Python files. For example, all database transactions and file uploads are processed in the Python *services.py* file. Thus, if a view triggers a file upload, the corresponding function can be found in *services.py*. In general, code blocks with database transactions have been set to atomic. If the whole code block is successfully completed, changes are committed to the database. If an exception is raised, the database transactions are rolled back and no changes are applied. Each Python file has its own distinct responsibility, which enables modularization and eases extension, debugging or modification.

3.2 Front end overview

If the user tries to access any of the URLs, the current login status is checked. If the user is not logged in, the user gets redirected to the login view. Prior to login the user needs to be registered, the registration view can be reached by pressing a “sign up” button on the login page. In general, if buttons or links are pressed by the user, requests are sent to the server. Requests are processed by view functions and depending on the request, the user gets forwarded to other views or redirected to the same view. After successful login, the user is forwarded to the blast_project view, the main page of the application. Based on this view all other views can be reached. Projects can be created by visiting the project creation view. After creation, project details (blast_project/project_id) can be displayed by pressing a “Project Details” button on the main page. By pressing the “Delete Project” button, the user will be forwarded to the project deletion view (blast_project/project_id/delete). The pipeline can get executed and monitored on the pipeline dashboard view (blast_project/project_id/pipeline_dashboard). Additional databases can be uploaded (genome_upload) and the user can check whether a species is contained in the nr database or not, by visiting the species taxonomic node view (species_taxid).

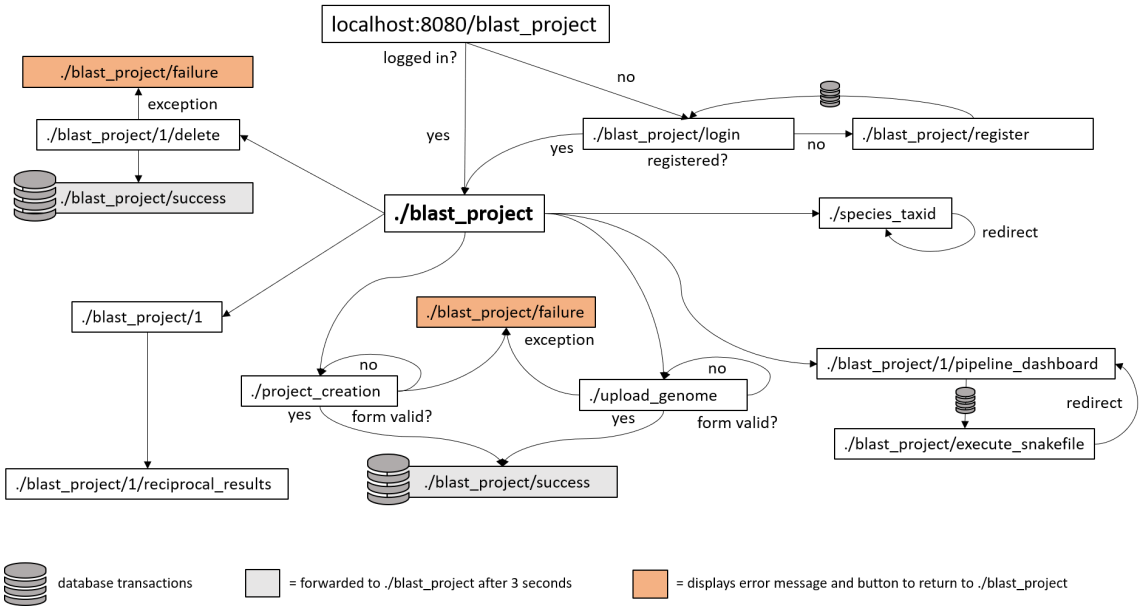

Figure 4: Front end overview. Available URLs are placed in rectangles. Database transactions are marked with a database symbol. Grey shaded rectangles and orange shaded rectangles are success and failure sites. The dots in front of the first backslash refer to the ip-address and port number of the host machine on which the application is running. The number one inside some of the URLs is a reference for a project_id.

Any requests that are processed by the server can end in a success or failure. A success is displayed via the success page, which redirects the user after three seconds to the main page, if an exception is raised a failure page with information regarding the exception is displayed.

3.3 The blast project view as main page

The main page was developed as an anchor point and dashboard for created projects. Based on this page, which is directly displayed after a successful login, the user is able to reach all relevant subsites, e.g. the project creation or project detail views.

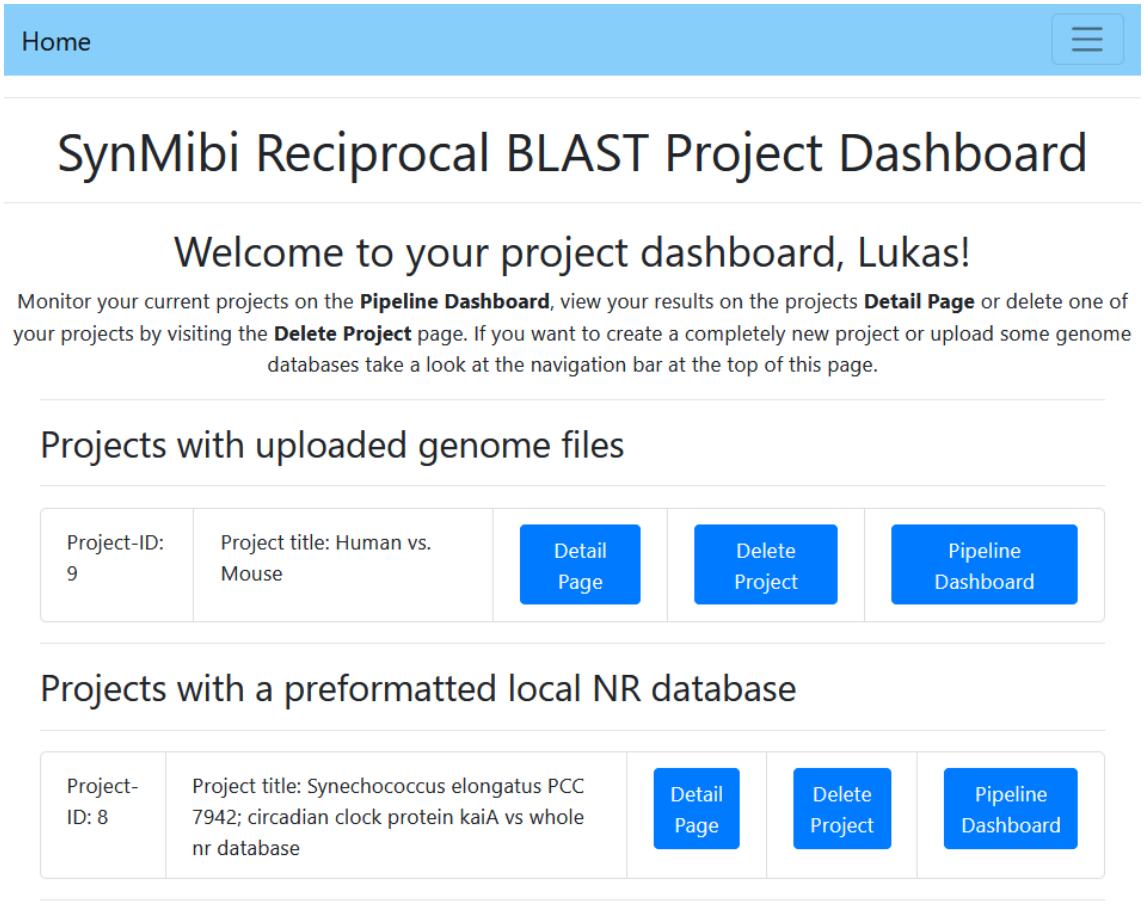

Figure 5: Main page and project dashboard. After a successful login, the user is forwarded to this view. Created projects are listed with buttons that can direct the user to other project relevant subsites.

Projects created by the currently logged in user are displayed on this page. The main page separates the two project types into two containers. Each container lists the corresponding projects with some additional information. For each project links to the project detail, project deletion and pipeline dashboard pages are displayed as blue buttons. By pressing those buttons the user is forwarded to the relevant view. In addition, the main page contains a navigation bar for accessing links to the species name checkup, the upload genome and project creation views.

3.4 Project creation interface for a structural project setup

The pipeline has to handle different workflow steps. The first step involves the input of query sequences and additional data such as the underlying genomes, taxonomic nodes, or BLAST settings. Deploying necessary data is crucial for executing the pipeline. For this reason a web

interface was developed that eases the correct data deployment. An automated project build process was implemented that prepares pipeline specific files after successful data validation. All necessary data for executing the pipeline is collected within the project creation page. Two different possibilities for project creation have been developed and can be accessed on the project creation view. The user can choose between those possibilities by pressing a radio button that is labelled with the relevant project type. The first option for creating a project is to select the project build and execution process based on a preformatted nr database (s. Fig. 8). For this project type the user has to specify scientific names for species in order to define the organisms for the forward and backward BLAST. If the user wants to analyse the query sequences against the whole nr database, no forward genome has to be specified. After submitting, the application tries to convert the scientific names into taxonomic nodes. If a scientific name is not present in the taxonomic database an error is thrown that is passed to the view. In addition to scientific names the user has to provide a query sequence file with protein sequences of the query species. The second option for creating a project is to upload genome files for the forward and backward BLAST (s. Fig. 9). Additionally this option allows the user to choose from previously uploaded files.

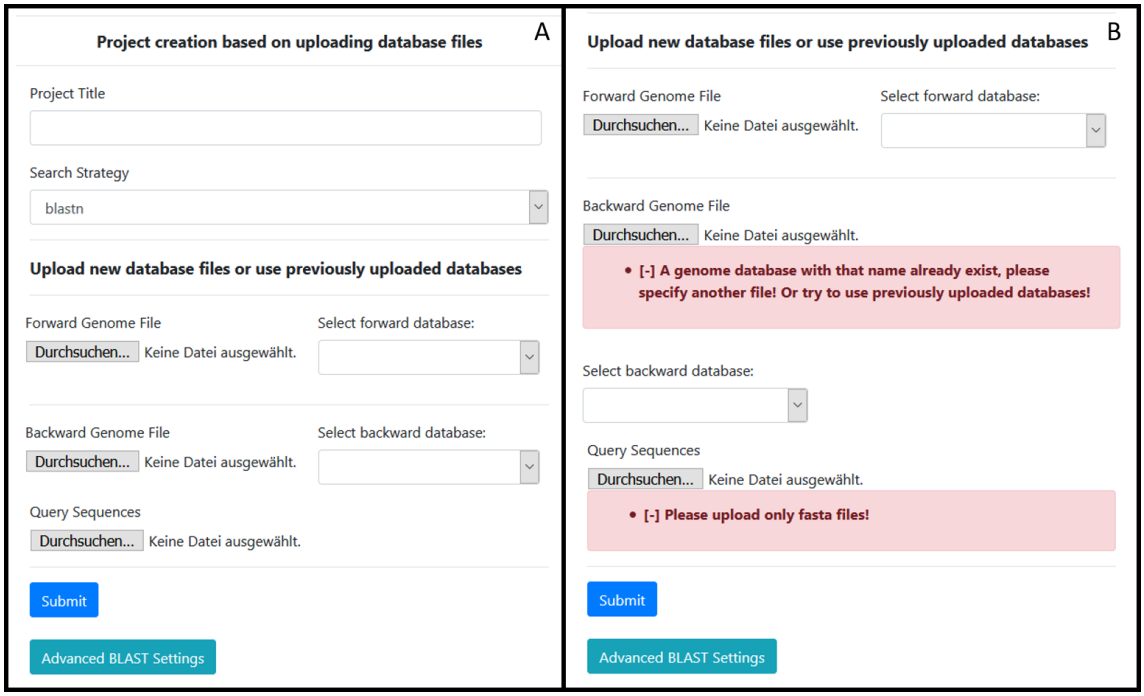

Figure 6: Section of the project creation page before data submitment (A) and after a failed submitment (B). Figure 6.B. shows possible form validation notifications for the query sequence and backward genome file.

All uploaded genome and query files as well as the query sequence file have to be in the FASTA format. For both project creation options, the forward and backward BLAST settings can get altered separately but default options are recommended. Albeit, depending on system resources the –thread option can get updated in order to boost the BLAST processes.

Errors that are thrown during validation of input data are passed to the view via project type specific form classes. Validation errors are stored in the form object and are accessed in the respective template files. Form validation errors are displayed under the relevant input fields with light-red rectangles and dark-red notes. If input data validation fails nothing will be stored in the database and no files are uploaded. This is done by setting the database transaction in the triggered view function for project creation to atomic. Thus, all database transactions are done at once and if any exception is raised nothing will be stored. In turn if input validation succeeds the project build process is triggered. The build process consists of several steps that are slightly different depending on the project type. In general, input data is written into the appropriate database tables and a project directory is created into which project-specific data is uploaded. A snakemake configuration file and, for nr based projects, files that contain taxonomic nodes are written into that directory during the build process.

3.5 Project details for informations and results

The view was developed and integrated into the application for displaying project specific data and relevant results. BLAST settings, genome and query file names as well as the project name and for nr based projects scientific names for the forward and backward BLAST are listed at the top of the page. The results of the pipeline are displayed in a table that can be viewed by pressing the reciprocal results table button. Furthermore, interactive result graphs are displayed on this page. These graphs differ depending on the project type and on the amount of RBHs that have been found. In projects with uploaded genome files, the graphs show statistical information based on reported e-values, bitscores, and the percentage of identity between RBHs as well as the amount of RBHs that have been found for each query sequence. For nr based projects taxonomic informations are available within the result tables and graphs. Hence, there is an additional interactive graph, which shows the amount of organisms in which RBHs have been inferred.

3.6 Pipeline dashboard for Snakemake execution and monitoring

Monitoring and execution of created projects is done with the pipeline dashboard views. The pipeline can get executed by pressing the button “execute snakemake”. By pressing the button the database field “executed” of the associated project is set to true and the user gets redirected to the pipeline dashboard (s. Fig. 4). If the pipeline has been executed, different monitoring possibilities are displayed. The user can decide whether to observe the current workflow on the pipeline dashboard or with some more information on the panoptes front end. The panoptes application allows monitoring of executed Snakemake workflows in real time. After pipeline execution a progress bar is displayed at the pipeline dashboard. The amount of displayed progress is managed by reading the latest Snakemake log file. In addition, the user can examine the whole content of the latest Snakemake log file by pressing the appropriate button.

3.7 Pipeline execution via Snakemake

Snakemake is used as the applications workflow management engine for executing and monitoring the reciprocal BLAST pipeline. Snakemake is executed with parameter flags. The *–directory* parameter is used to ensure that Snakemake uses the project directory as working directory. Snakemakes output files such as the log files and result files are written into that directory. The project specific configuration file is loaded into the snakefile via the *–configfile* parameter. An additional monitoring option is enabled with panoptes. The panoptes server is started with the use of Snakemakes *–wms-monitor* parameter. Snakemake is executed with the *–cores 2* flag. The non-file rule parameter "params" is used for the forward and backward BLAST rules in order to deliver user defined or default BLAST settings to the invocation of BLAST.

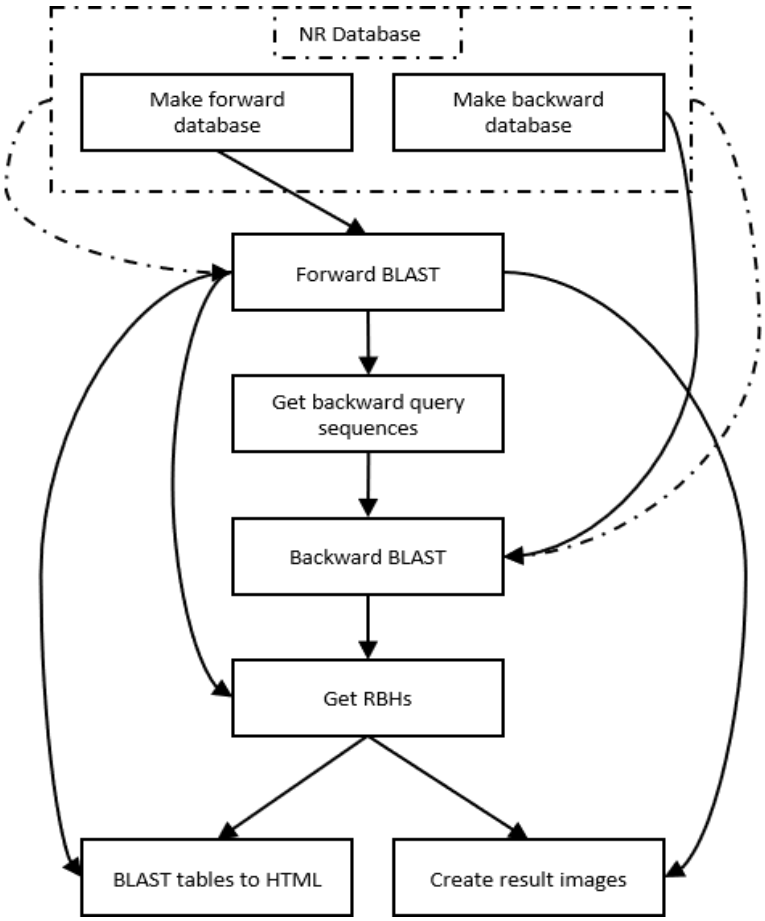

Figure 7: Snakemake directed acyclic graph for pipeline execution. The dashed rectangle symbolizes the previously conducted process of nr database formatting. For nr based project types the first two rules are not executed. Arrows symbolize output files that are used as input files for the respective rules.

Two snakefiles, one for each project type, have been developed that do serve as the central pipeline execution points. Both snakefiles contain all relevant rules for pipeline execution. For projects that use uploaded genomes two rules have been created that execute the *makeblastdb* formatting procedure for the uploaded genome FASTA files into BLAST databases

(rules: “Make forward database” and “Make backward database”). For projects based on the nr database those two rules have been excluded. The workflow process for both snakefiles is managed by a build-target rule. This rule takes the output files from the last two rules as input files. Snakemake then attempts to access these files by building a DAG for rule execution orders, which follows the build process of output files to the first rules of the pipeline that can get executed. Those rules are the “Make forward database” and “Make backward database” for none nr projects. For uploaded genome file projects the forward BLAST is executed after database formatting. Projects with the nr database can directly start with the execution of the forward BLAST. BLAST searches within the nr database are limited by taxonomy with the *-taxidlist* option of the BLAST command-line tool. The forward BLAST outputs a csv file that serves as input for the rule that executes the extraction of RBHs, for the last two result processing rules, and for the next rule whose task is to extract query sequences for the backward BLAST. This extraction is done via custom Python scripts that do collect all accession ids from the subjects of the forward BLAST output, in combination with the *blastdbcmd* program that can extract sequences out of BLAST databases on the basis of accession identifiers. The next executed rule is the backward BLAST. The default settings of the backward BLAST are designed to limit the output to the best alignment partner of the query sequences. This is done with the *-num_alignment* flag of the BLAST command-line tool, which is set to one. The output of the backward BLAST is another csv file. This file, together with the csv file from the forward BLAST, is the input for the last necessary pipeline step for inferring orthologs. During this step the csv files are processed and two dictionaries are created. The keys of these dictionaries are the query identifier and the values are subject identifiers. With this definition the query keys for the forward BLAST dictionary are the subject values of the backward BLAST dictionary and query keys of the backward BLAST dictionary are subject values of the forward BLAST dictionary. An orthologous sequence is found if there is a simultaneous match of key, value pairs of the forward dictionary with value, key pairs of the backward dictionary. Particularly, if one value of a key of the forward dictionary matches a key of the backward dictionary and the corresponding value matches the key of the forward dictionary an RBH is identified. The orthologous sequences are written into a text file that serves as input for the result processing rules. In combination with the forward BLAST output, result processing with report and graph creation is done as the last step of the reciprocal BLAST pipeline. The rule “BLAST tables to HTML” is in both projects equivalent. The rule produces a HTML table that displays RBHs and additional information of the BLAST run. The rule “Create result images” has been created in order to produce graphs. Those graphs differ between the two project types and the amount of query sequences.

3.8 Comparison to previously used reciprocal BLAST methods

There are two steps for executing a reciprocal BLAST with the developed application. First, the user has to create a project on the project creation front end, either based on the nr database or on custom genome databases, that are uploaded as FASTA files. Secondly, the user has to press the execute button on the pipeline dashboard front end. Previously, a reciprocal BLAST was conducted via remote searches on the NCBI BLAST web-interface with a subsequent processing of result data. These steps involved the manual execution of functions, which is error prone and time-consuming. Furthermore, since September 8, 2020, NCBI limits remote BLAST searches, which complicated reciprocal BLAST analyses on whole genomes.

3.9 Inference of orthologous sequences of the circadian clock gene kaiA from *Synechococcus elongatus* PCC 7942

In order to test the pipeline integrity it was executed with the circadian clock gene kaiA as input query sequence. BLAST default settings were used during project creation. The reciprocal BLAST analysis was conducted against the whole nr database. The workflow procedure was executed inside a docker container on a custom notebook, an Acer Aspire E5-575G-527E. The forward BLAST took three hours, 45 minutes and 43.13 seconds and resulted in 990 unique homologous sequences. Those homologous sequences served as query sequences for the backward BLAST against the *Synechococcus elongatus* PCC 7941 genome (of the nr database). The backward BLAST took two minutes and 15.28 seconds to complete. 990 unique query sequences matched against three unique *Synechococcus elongatus* PCC 7941 sequences. The inference of RBHs took 3.23 seconds and resulted in 606 RBHs. 902 unique species have RBHs with the kaiA protein of *Synechococcus elongatus* PCC 7941.

This analysis was compared to a previously conducted reciprocal BLAST of kaiA against a database, which contained all genomes from the genbank protein database labelled as “Complete Genome” or “Chromosome” (Schmelling et al. 2017). In the previous reciprocal BLAST analysis, 72 unique orthologous protein sequences have been identified. 35 of those 72 accession identifiers (accessions) are not inside the new identified set of RBHs. Of those 35 accessions, 25 accessions are not part of the nr database because their identifiers have changed, thus the underlying sequences of those accessions are in the database. The ten other sequences are inside the forward BLAST output but they do not match against the kaiA query sequence in the backward BLAST, moreover they match against the 4G86_A sequence, which is labelled as “cofactor DBMIB bound to the full length circadian clock protein kaiA”. 4G86_A is a sequence of the Protein Data Bank, which is a database for 3D structural forms of proteins. The 25 renamed accession identifiers are inside the forward BLAST output, 15 of them do match against the kaiA sequence, the other ten sequences do match against 4G86_A.

4 Discussion

4.1 Snakemake enhances reciprocal BLAST analyses

The integration of Snakemake as the core program for pipeline execution greatly enhances and eases the reproducibility, code structure, and monitoring process of the reciprocal BLAST pipeline. Due to clear structuring and parameter compositions the snakefiles allow a clear and readable workflow setup. This is true, especially when compared to the previously existing scripts for conducting a reciprocal BLAST. One rule is present for every workflow step, within those rules all necessary parameters, commands, or executables are listed. This eases the understanding of the reciprocal BLAST pipeline process, which in turn allows an easy and fast altering or editing of rules if necessary. Additional post-processing steps such as the direct computation of phylogenetic trees, annotation procedures, or comparisons with orthologous sequence databases are easy to realize. This simply involves additional rules with required parameters and an altering of the build-target rule. Due to the utilization of project specific configuration files, which are used by the snakefile it is possible to easily add parameters for external programs. Furthermore, this enables reproducibility and the comparison with different databases, thus, project settings can be reused by simply copying the project directories or if the same settings should be used for other query sequences by copying the configuration file. In addition, pipeline monitoring and execution time analyses are enabled due to Snakemakes log files that are written into target project directories and the –wms-monitor flag, which initiates a communication between Snakemake and the panoptes application. Panoptes allows the monitoring of executed workflows in real time and it lists time frames for each finished job. In comparison with previous methods 3.8 for conducting reciprocal BLAST analyses, the utilization of Snakemake greatly enhances the ease of performing such analyses.

4.2 Snakemake allows further enhancements

Nevertheless, there are still some enhancements available that Snakemake offers. The fundamental reciprocal BLAST pipeline consists of five steps (1.6). Currently, two snakefiles have been developed and based on these five core steps, the snakefiles inherit some similar rules. The decision to create two snakefiles has been made due to the process of post-processing and database creation. Thus, projects executed with the preformatted nr database have access to taxonomic information and there is no need to execute any database formatting procedure. However, it would be possible to merge both snakefiles due to Snakemakes rule execution process. Rules that are not needed by the current project are skipped. E.g. if a database is already formatted, the target formatting rule wont be executed. Further enhancements can get implemented with utilization of additional parameters. The pipeline execution via Snakemake is done within the developed web application, this gives rise to utilization of Snakemakes report parameter. The report parameter automatically creates HTML reports, which can be accessed and displayed in combination with the web tool. The –cores parameter of Snakemakes

command-line API allows the assignment of more than one core. It would be applicable to change the number of cores depending on the available system.

4.3 The web application eases data deployment and pipeline execution

The integration of Snakemake in a web application further enhances and eases the execution of reciprocal BLAST analyses. The developed views enable project creation, execution and monitoring. Similar to the public available BLAST online tool the reciprocal BLAST analyses begins with the creation of projects. This is a crucial step as the input data serves as basis for conducting the analyses. Hence, it is important to offer a reliable and easy-to-use platform for project setups. This is achieved by utilization of Django's MVT framework. The project creation view has a clear structure that is self explaining and additional help is provided by form validation. If the user forgets to submit necessary or enters erroneous data, form validation will raise a validation error, which will be present in the form object. In this case, the user gets redirected to the project creation view, that now displays the error message under the respective field. Furthermore, no data is stored in the database or is uploaded on the server. If the form is fulfilled correctly, the submitment will render a success view that redirects the user to the main view, where the newly created project is listed. This browser based data input, together with validation and error handling optimizes the procedure of data deployment.

The project dashboard is designed for displaying projects and relevant links to subpages. Projects are only listed for the current user, which enables a clear overview and the partition to projects from other users. Albeit, the developed web application greatly enhances the execution of reciprocal BLAST analyses, there are still possibilities to further optimize processes. E.g. the user has to download query sequences manually, not all BLAST settings are modifiable and the formatting procedure of BLAST databases is done without taxonomic information. Furthermore, if a workflow for a certain project gets executed and results in an error, the user has to delete the project, hence altering and updating procedures are not yet available. Nevertheless, it is possible to analyse the error in the front end due to attachment of Snakemakes last log file in the pipeline dashboard and a lot of error handling is done previously, during input data validation.

In comparison to previously conducted reciprocal BLAST analyses, the developed application greatly eases the execution and reproducibility. The forward and backward BLAST analyses do not have to be executed manually on the NCBI web interface. This enables the possibility for conducting whole genome reciprocal BLAST analyses. Even on custom notebooks reciprocal BLAST analyses can easily be done (3.9). For the correct pipeline process existing functions do not have to be altered. Once the user has uploaded and submitted all necessary data, the setup is done. Furthermore, utilization of docker allows an application deployment on every operating system. The installation process requires a bunch of different Python packages and

the BLAST executables. Thus, docker eases the installation as it automatically downloads and installs packages and the required software. However, due to volume sharing the BLAST analyses appear to be slower, but depending on the computational time of the pipeline this effect might be negligible (Di Tommaso et al. 2015). In general, analyses with numerous query sequences are slower on systems with less computational power. Thus, it is necessary to deploy the application on a system with greater computational power if whole genome reciprocal BLAST analyses should be conducted.

There are a lot of gene databases available with different levels of completeness (Sayers et al. 2018). Scientific projects concerning sequence analyses highly depend on those databases. For most projects it is important to have distinct and organism specific genome databases available. This can shorten the computational time needed for sequence analyses and enhance the accuracy for diverse approaches, such as the identification of orthologous sequences. Thus, a broad spectrum of accessible genome databases is preferable. Since previous BLAST analyses were carried out on NCBI servers, all available databases could be used, this is not true for the developed application. Currently, the application is limited by the utilization of the nr database and by uploading genomes via FASTA files. Nevertheless, databases can easily be added due to the clear structure and modularization of the application.

4.4 Outlook

The current application can get further enhanced. Execution of Snakemake with the Popen function of the subprocess module of Python opens a new process. It is possible to access the current status of the process with the communicate function. It would be useful to integrate such process monitoring and direct process communication as it is currently not possible to terminate a pipeline without application shutdown. In addition, information obtainable by communication with the spawned process can be used to optimize error handling. Currently, there is only one pre-formatted database available, which is not optimal, which was observed when orthologues were inferred for the circadian clock gene kaiA from *Synechococcus elongatus*. Some orthologous sequences, that have been identified in the previous analysis (Schmelling et al. 2017), matched against a sequence which is part of the Protein Data Bank. Without the 4G86_A sequence, all of the 72 sequences would have been found. In order to improve accuracy of ortholog inference, other databases with a lower level of redundancy must be integrated, such as the combined genbank genomes labelled as “Complete Genome and “Chromosome, that have been used in the analysis conducted in Schmelling et al. 2017. Other sequences have been renamed, thus they reside with their new accession identifier in the nr database. This problem can be solved by searching for query sequences in the chosen database with the blastdbcmd program of the BLAST command-line tool. If sequences are not inside the database a warning with the respective accession identifier can be displayed. Furthermore, database formatting rules can get extended by the taxid or taxid_map parameter of the *makeblastdb* program. With these parameters it is possible to integrate taxonomic information inside uploaded databases. Thus, the output format of genome upload projects can get adjusted to the output format of nr projects.

However, the reciprocal BLAST analysis is important for first insights of function and phylogeny of newly sequenced genes. The web application in combination with Snakemake greatly enhances the utilization and applicability of reciprocal BLAST analyses. Furthermore, the intuitive usability of the developed application enables an easy-to-use execution of reciprocal BLAST analysis for scientists who are not familiar with programming. Further development and integration of other databases, software tools and analysis options can optimize and extend the application.

References

[1] William R. Pearson. “An introduction to sequence similarity ("homology") searching”. In: *Current protocols in bioinformatics* Chapter 3 (2013). URL: http://dx.doi.org/10.1002/0471250953.bi0301s42.

[2] Gerald R. Reeck, Christoph de Haën, David C. Teller, Russell F. Doolittle, Walter M. Fitch, Richard E. Dickerson, Pierre Chambon, Andrew D. McLachlan, Emanuel Margoliash, Thomas H. Jukes, and Emile Zuckerkandl. “Homology in proteins and nucleic acids: A terminology muddle and a way out of it”. In: *Cell* 50.5 (1987), p. 667. ISSN: 0092-8674. DOI: https://doi.org/10.1016/0092-8674(87)90322-9. URL: https://www.sciencedirect.com/science/article/pii/0092867487903229.

[3] Eugene Koonin. “Orthologs, paralogs, and evolutionary genomics”. In: *Annual Review of Genetics* 39 (2005), pp. 309–338. ISSN: 00664197. DOI: 10.1146/annurev.genet.39.073003.114725.

[4] William R. Pearson. “Empirical statistical estimates for sequence similarity searches”. In: *Journal of Molecular Biology* 276.1 (1998), pp. 71–84. ISSN: 00222836. DOI: 10.1006/jmbi.1997.1525.

[5] Like Fokkens, Sandra M.C. Botelho, Jos Boekhorst, and Berend Snel. “Enrichment of homologs in insignificant BLAST hits by co-complex network alignment”. In: *BMC Bioinformatics* 11 (2010). ISSN: 14712105. DOI: 10.1186/1471-2105-11-86.

[6] John A. Gerlt and Patricia C. Babbitt. “Can sequence determine function?” In: *Genome Biology* 1.5 (2000), pp. 1–10. ISSN: 14656906. DOI: 10.1186/gb-2000-1-5-reviews0005.

[7] Sonia Facchin, Raffaele Lopreiato, Maria Ruzzene, Oriano Marin, Geppo Sartori, Claudia Götz, Mathias Montenarh, Giovanna Carignani, and Lorenzo A Pinna. “Functional homology between yeast piD261/Bud32 and human PRPK: both phosphorylate p53 and PRPK partially complements piD261/Bud32 deficiency”. In: *FEBS Letters* 549.1 (2003), pp. 63–66. ISSN: 0014-5793. DOI: https://doi.org/10.1016/S0014-5793(03)00770-1. URL: https://www.sciencedirect.com/science/article/pii/S0014579303007701.

[8] Fredj Tekaia. “Inferring orthologs: Open questions and perspectives”. In: *Genomics Insights* 9 (2016), pp. 17–28. ISSN: 11786310. DOI: 10.4137/GEI.S37925.

[9] Walter M. Fitch. “Distinguishing homologous from analogous proteins”. In: *Systematic Zoology* 19.2 (1970), pp. 99–113. ISSN: 00397989. DOI: 10.2307/2412448.

[10] Jianzhi Zhang. “Evolution by gene duplication: An update”. In: *Trends in Ecology and Evolution* 18.6 (2003), pp. 292–298. ISSN: 01695347. DOI: 10.1016/S0169-5347(03)00033-8.

[11] Adrian M. Altenhoff, Romain A. Studer, Marc Robinson-Rechavi, and Christophe Dessimoz. “Resolving the Ortholog Conjecture: Orthologs Tend to Be Weakly, but Significantly, More Similar in Function than Paralogs”. In: *PLOS Computational Biology* 8.5 (May 2012), pp. 1–10. DOI: 10.1371/journal.pcbi.1002514. URL: https://doi.org/10.1371/journal.pcbi.1002514.

[12] Erik L.L. Sonnhammer and Eugene Koonin. “Orthology, paralogy and proposed classification for paralog subtypes”. In: *Trends in Genetics* 18.12 (2002), pp. 619–620. ISSN: 0168-9525. DOI: https://doi.org/10.1016/S0168-9525(02)02793-2. URL: https://www.sciencedirect.com/science/article/pii/S0168952502027932.

[13] Maido Remm, Christian E.V. Storm, and Erik L.L. Sonnhammer. “Automatic clustering of orthologs and in-paralogs from pairwise species comparisons”. In: *Journal of Molecular Biology* 314.5 (2001), pp. 1041–1052. ISSN: 00222836. DOI: 10.1006/jmbi.2000.5197.

[14] Nan Song, Jacob M. Joseph, George B. Davis, and Dannie Durand. “Sequence similarity network reveals common ancestry of multidomain proteins”. In: *PLoS Computational Biology* 4.5 (2008). ISSN: 1553734X. DOI: 10.1371/journal.pcbi.1000063.

[15] Eugene Koonin, Kira S. Makarova, and L. Aravind. “Horizontal Gene Transfer in Prokaryotes: Quantification and Classification”. In: *Annual Review of Microbiology* 55.1 (2001). PMID: 11544372, pp. 709–742. DOI: 10.1146/annurev.micro.55.1.709. eprint: https://doi.org/10.1146/annurev.micro.55.1.709. URL: https://doi.org/10.1146/annurev.micro.55.1.709.

[16] David M. Kristensen, Yuri I. Wolf, Arcady R. Mushegian, and Eugene Koonin. “Computational methods for Gene Orthology inference”. In: *Briefings in Bioinformatics* 12.5 (2011), pp. 379–391. ISSN: 14675463. DOI: 10.1093/bib/bbr030.

[17] Boris Mirkin, Ilya Muchnik, and Temple F. Smith. “A Biologically Consistent Model for Comparing Molecular Phylogenies”. In: *Journal of Computational Biology* 2.4 (1995). PMID: 8634901, pp. 493–507. DOI: 10.1089/cmb.1995.2.493. eprint: https://doi.org/10.1089/cmb.1995.2.493. URL: https://doi.org/10.1089/cmb.1995.2.493.

[18] Roderic D.M. Page and Michael A. Charleston. “From Gene to Organismal Phylogeny: Reconciled Trees and the Gene Tree/Species Tree Problem”. In: *Molecular Phylogenetics and Evolution* 7.2 (1997), pp. 231–240. ISSN: 1055-7903. DOI: https://doi.org/10.1006/mpev.1996.0390. URL: https://www.sciencedirect.com/science/article/pii/S1055790396903905.

[19] Heng Li et al. “TreeFam: a curated database of phylogenetic trees of animal gene families.” In: *Nucleic acids research* 34.Database issue (2006), pp. 10–16. ISSN: 13624962. DOI: 10.1093/nar/gkj118.

[20] Jaime Huerta-Cepas, Salvador Capella-Gutierrez, Leszek P. Pryszcz, Ivan Denisov, Diego Kormes, Marina Marcet-Houben, and Toni Gabaldón. "PhylomeDB v3.0: An expanding repository of genome-wide collections of trees, alignments and phylogeny-based orthology and paralogy predictions". In: *Nucleic Acids Research* 39.SUPPL. 1 (2011). ISSN: 03051048. DOI: 10.1093/nar/gkq1109.

[21] Roderic D.M. Page and Michael A. Charleston. "From Gene to Organismal Phylogeny: Reconciled Trees and the Gene Tree/Species Tree Problem". In: *Molecular Phylogenetics and Evolution* 7.2 (1997), pp. 231–240. ISSN: 1055-7903. DOI: https://doi.org/10.1006/mpev.1996.0390. URL: https://www.sciencedirect.com/science/article/pii/S1055790396903905.

[22] W. Ford Doolittle. "Phylogenetic classification and the universal tree". In: *Science* 284.5423 (1999), pp. 2124–2128. ISSN: 00368075. DOI: 10.1126/science.284.5423.2124.

[23] Roman L. Tatusov, Eugene Koonin, and David J. Lipman. "A genomic perspective on protein families". In: *Science* 278.5338 (1997), pp. 631–637. ISSN: 00368075. DOI: 10.1126/science.278.5338.631.

[24] Saul B. Needleman and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of Molecular Biology* 48.3 (1970), pp. 443–453. ISSN: 0022-2836. DOI: https://doi.org/10.1016/0022-2836(70)90057-4. URL: https://www.sciencedirect.com/science/article/pii/0022283670900574.

[25] Michael S. Waterman. "General methods of sequence comparison". In: *Bulletin of Mathematical Biology* 46.4 (1984), pp. 473–500. ISSN: 0092-8240. DOI: https://doi.org/10.1016/S0092-8240(84)80054-3. URL: https://www.sciencedirect.com/science/article/pii/S0092824084800543.

[26] William R. Pearson and Webb Miller. "Dynamic programming algorithms for biological sequence comparison". In: Methods in Enzymology 210 (1992), pp. 575–601. ISSN: 0076-6879. DOI: https://doi.org/10.1016/0076-6879(92)10029-D. URL: https://www.sciencedirect.com/science/article/pii/007668799210029D.

[27] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. "Basic local alignment search tool". In: *Journal of Molecular Biology* 215.3 (1990), pp. 403–410. ISSN: 00222836. DOI: 10.1016/S0022-2836(05)80360-2.

[28] S. Karlin and Stephen F. Altschul. "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes". In: *Proceedings of the National Academy of Sciences* 87.6 (1990), pp. 2264–2268. ISSN: 0027-8424. eprint: https://www.pnas.org/content/87/6/2264.full.pdf. URL: https://www.pnas.org/content/87/6/2264.

[29] Stephen F. Altschul, Thomas Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs”. In: *Nucleic Acids Research* 25.17 (Sept. 1997), pp. 3389–3402. ISSN: 0305-1048. DOI: 10.1093/nar/25.17.3389. eprint: https://academic.oup.com/nar/article-pdf/25/17/3389/3639509/25-17-3389.pdf. URL: https://doi.org/10.1093/nar/25.17.3389.

[30] Christiam Camacho, Vahram Avagyan, Ning Ma, George Coulouris, Jason Papadopoulos, Kevin Bealer, and Thomas Madden. “BLAST+: architecture and applications”. In: *BMC Bioinformatics* 10.1 (2009), p. 421. ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-421. URL: http://www.biomedcentral.com/1471-2105/10/421.

[31] Liisa B. Koski and G. Brian Golding. “The closest BLAST hit is often not the nearest neighbor”. In: *Journal of Molecular Evolution* 52.6 (2001), pp. 540–542. ISSN: 00222844. DOI: 10.1007/s002390010184.

[32] Ioanna Karamichali, V. Lila Koumandou, Amalia D. Karagouni, and Sophia Kossida. “Frequent gene fissions associated with human pathogenic bacteria”. In: *Genomics* 103.1 (2014), pp. 65–75. ISSN: 0888-7543. DOI: https://doi.org/10.1016/j.ygeno.2014.02.001. URL: https://www.sciencedirect.com/science/article/pii/S088875431400007X.

[33] Christopher S. Henry et al. “Systematic identification and analysis of frequent gene fusion events in metabolic pathways”. In: *BMC Genomics* 17.1 (2016). ISSN: 14712164. DOI: 10.1186/s12864-016-2782-3. URL: http://dx.doi.org/10.1186/s12864-016-2782-3.

[34] Roman L. Tatusov, Arcady R. Mushegian, Peer Bork, Nigel P. Brown, William S. Hayes, Mark Borodovsky, Kenneth E. Rudd, and Eugene Koonin. “Metabolism and evolution of Haemophilus influenzae deduced from a whole-genome comparison with Escherichia coli”. In: *Current Biology* 6.3 (1996), pp. 279–291. ISSN: 0960-9822. DOI: https://doi.org/10.1016/S0960-9822(02)00478-5. URL: https://www.sciencedirect.com/science/article/pii/S0960982202004785.

[35] Scott McGinnis and Thomas Madden. “BLAST: at the core of a powerful and diverse set of sequence analysis tools”. In: *Nucleic Acids Research* 32.suppl$_2$ (July 2004), W20–W25. ISSN: 0305-1048. DOI: 10.1093/nar/gkh435. eprint: https://academic.oup.com/nar/article-pdf/32/suppl_2/W20/6210425/gkh435.pdf. URL: https://doi.org/10.1093/nar/gkh435.

[36] Johannes Köster and Sven Rahmann. “Snakemake-a scalable bioinformatics workflow engine”. In: *Bioinformatics* 28.19 (2012), pp. 2520–2522. ISSN: 14602059. DOI: 10.1093/bioinformatics/bts480.

[37] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[38] *Anaconda Software Distribution*. Version Vers. 2-2.4.0. 2020. URL: https://docs.anaconda.com/.

[39] Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11 (Mar. 2009), pp. 1422–1423. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp163. eprint: https://academic.oup.com/bioinformatics/article-pdf/25/11/1422/944180/btp163.pdf. URL: https://doi.org/10.1093/bioinformatics/btp163.

[40] Christiam Camacho, Thomas Madden, Tao Tao, Richa Agarwala, and Aleksandr Morgulis. "BLAST Command Line Applications User Manual". In: *Bethesda (MD): National Center for Biotechnology Information (US)* MD (2019), pp. 1–28. URL: https://www.ncbi.nlm.nih.gov/books/NBK279690/.

[41] Jonathan Kans. "Entrez Direct: E-utilities on the UNIX Command Line". In: *National Center for Biotechnology Information* Md (2020), p. 161. URL: https://www.ncbi.nlm.nih.gov/books/NBK179288/.

[42] *Django*. Version 2.2.5. 2019. URL: https://djangoproject.com.

[43] Richard D Hipp. *SQLite*. Version 3.31.1. 2020. URL: https://www.sqlite.org/index.html.

[44] PostgreSQL. *Documentation PostgreSQL 13.2*. Version 13.2. 2021. URL: https://www.postgresql.org/docs/release/13.2/.

[45] Dirk Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux journal* 2014.239 (2014), p. 2.

[46] *Bootstrap 4*. Version 4.6.x. 2020. URL: https://getbootstrap.com/.

[47] Nicolas M. Schmelling, Robert Lehmann, Paushali Chaudhury, Christian Beck, Sonja Verena Albers, Ilka M. Axmann, and Anika Wiegard. "Minimal tool set for a prokaryotic circadian clock". In: *BMC Evolutionary Biology* 17.1 (2017). ISSN: 14712148. DOI: 10.1186/s12862-017-0999-7.

[48] Paolo Di Tommaso, Emilio Palumbo, Maria Chatzou, Pablo Prieto, Michael L. Heuer, and Cedric Notredame. "The impact of Docker containers on the performance of genomic pipelines". In: *PeerJ* 2015.9 (2015), pp. 1–10. ISSN: 21678359. DOI: 10.7717/peerj.1273.

[49] Eric W Sayers et al. "Database resources of the National Center for Biotechnology Information". In: *Nucleic Acids Research* 47.D1 (Nov. 2018), pp. D23–D28. ISSN: 0305-1048. DOI: 10.1093/nar/gky1069. eprint: https://academic.oup.com/nar/article-pdf/47/D1/D23/27437595/gky1069.pdf. URL: https://doi.org/10.1093/nar/gky1069.

A Abbreviations

List of Abbreviations

DNA	Deoxyribonucleic acid
nr	non-redundant
HGT	Horizontal Gene Transfer
LCA	Last Common Ancestor
RBH	Reciprocal Best Hit
DPA	Dynamic Programming Algorithm
HSP	High Scoring Pairs
DAG	Directed Acyclic Graph
URL	Uniform Resource Locator
DOI	Digital Object Identifier
SQL	Structured Query Language
MVT	Model-View-Template
BLAST	Basic Local Alignment Search Tool
FTP	File Transfer Protocol
NCBI	National Center for Biotechnology Information
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
CSV	Comma Separated Values
XML	Extensible Markup Language
CSS	Cascading Style Sheets

B Appendix

blastp	Protein sequence similarity search against protein subjects or databases.
blastn	Nucleotide sequence similarity search against nucleotide subjects or databases.
blastx	Translates a nucloeitde query to a proetin sequence and conducts a similarity search against protein subjects or databases.
tblastx	Similarity search of a translated nucleotide query against translated nucleotide subjects or databases.
tblastn	Similarity search of a protein query against nucleotide subject sequences or databases translated at search time.
makeblastdb	Program for formatting BLAST databases.
blastdb_aliastool	Program for connecting different databases.
blastdbcmd	Report producing program for BLAST databases.
Scripts	Various other scripts that help with the utilization of NCBI databases or the execution of BLAST command-line tool programs.

Table 1: Overview of BLAST command-line tool programs

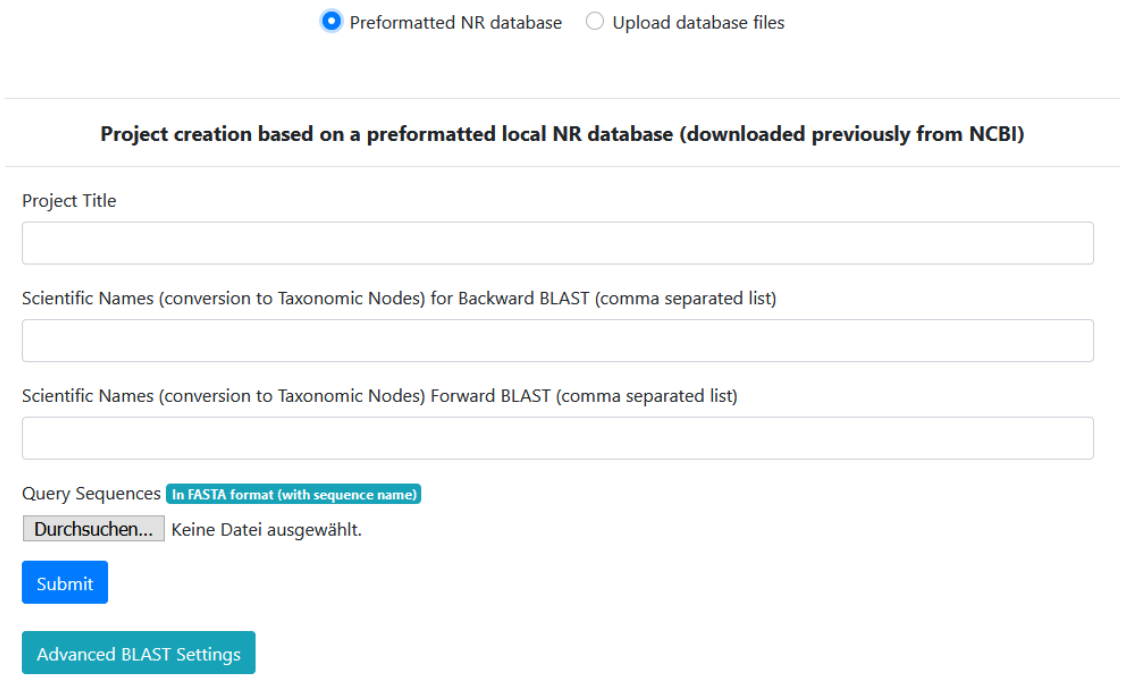

Figure 8: Section of the project creation view with the pre-formatted nr database. Scientific names are converted to taxonomic nodes via BioPython and the *get_species_taxid.sh* script provided by the BLAST C++ command-line tool. Taxonomic nodes are used for limiting BLAST searches.

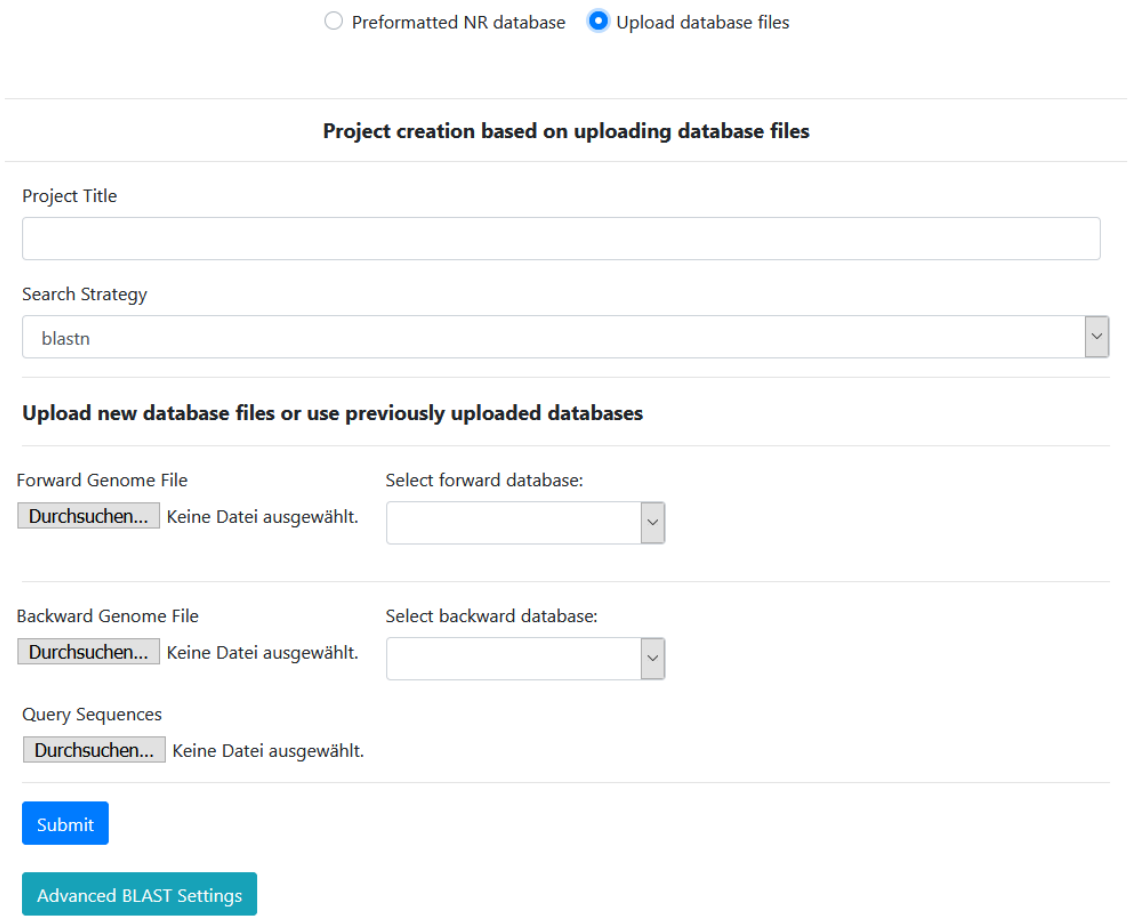

Figure 9: Section of the project creation view with uploaded genomes. Previously uploaded genomes can also be assigned as databases.

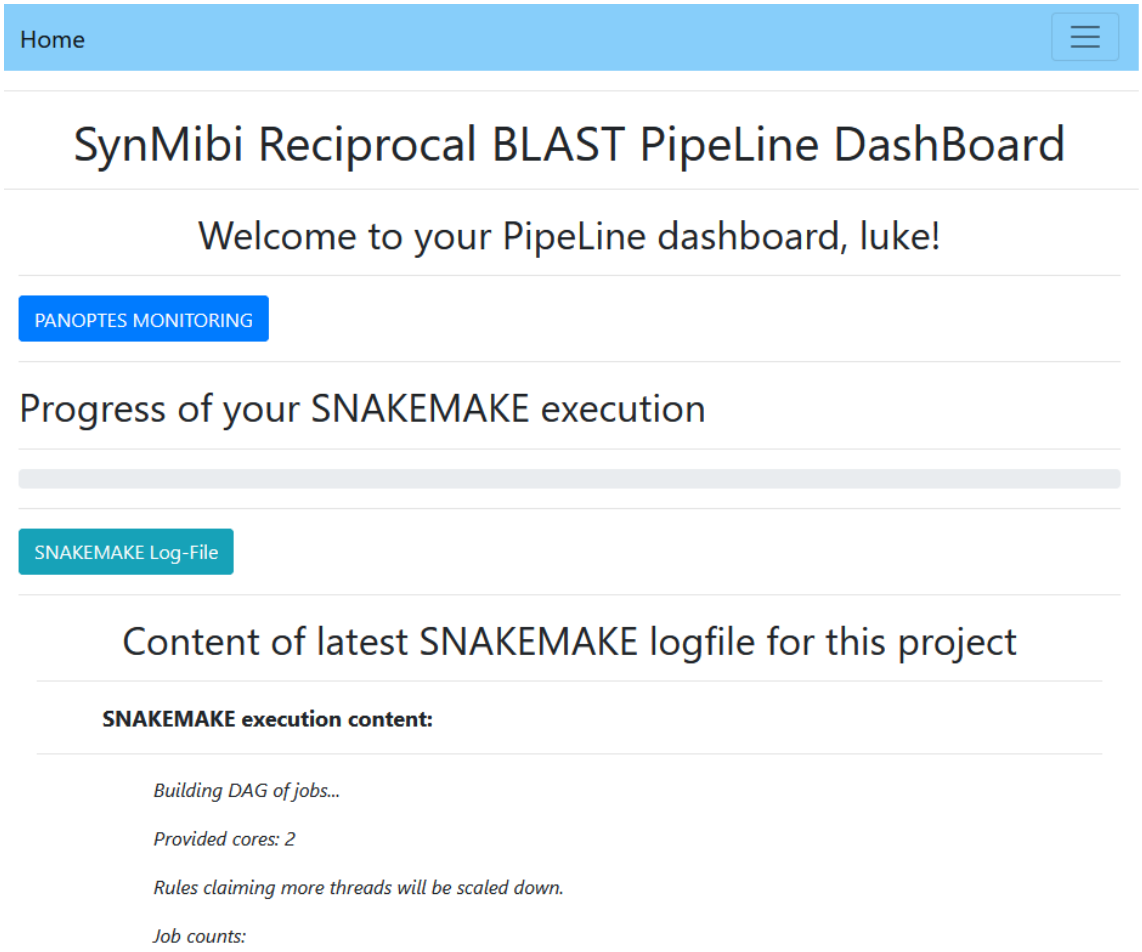

Figure 10: Section of the pipeline dashboard view directly after execution. No rule has finished, the progress bar is at zero percent.