Reproducing Analysis of StartLink

Karl Gemayel

August 28, 2020

Contents

1 Introduction

This document serves as a step-by-step instruction manual on how to replicate results from the StartLink paper. We note that many of the experiments are time-consuming and/or resource hungry. Therefore, we also provide raw results (from which graphs can be generated) as well as pre-constructed databases for the clades mentioned in the StartLink paper.

2 Downloading

To download, simply clone the git repository

```
git clone ...
```

3 Installation

3.1 StartLink

Once downloaded, binaries can be "installed" by running

```
source config.sh    # sets up environment variables
source install.sh   # creates binaries
```

The `config.sh` script loads environment variables (see script for details). These will help you navigate and run commands. It is essential that this is executed if you want to follow the steps listed below.

The `install.sh` script creates binary files for all `python` drivers. This makes them easy to execute. If preferred, drivers can be executed by calling the `python` interpreter directly. The methods of execution are then:

```
$bin/NAME_OF_DRIVER_py.sh                # if using installed binaries
python $driverpython/NAME_OF_DRIVER.py   # if using python interpreter
```

where `$driverpython` is loaded by the `config.sh` script.

3.1.1 Loading Python Libraries

Most of StartLink is built in Python. We use `conda` to manage package installations. Simply run

```
conda env create -f install/startlink.yml --name startlink  # needs to be executed only
    once
conda activate startlink                    # run whenever you start a new shell
```

3.2 External Tools

StartLink and StartLink+ and their analysis rely on a handful of external tools. The following need to be installed:

- GeneMarkS-2
  - Used for building StartLink+ predictions, and for analysis
  - Place installation in `$base/bin_external/gms2/`
  - Link: `http://exon.gatech.edu/GeneMark/license_download.cgi`
- ClustalO:
  - Used for constructing multiple sequence alignments
  - Install and must be accessible from `$PATH`
  - Link: `http://www.clustal.org/omega/#Download`
  - Install via Anaconda: `conda install -c bioconda clustalo`
- Diamond BLAST:
  - Used for generating target databases and finding orthologs
  - Install and must be accessible from `$PATH`
  - Link: `https://github.com/bbuchfink/diamond`
  - Install via Anaconda (recommended): `conda install -c bioconda diamond`
- Prodigal:
  - Used for initial analysis of gene-start prediction status
  - Place installation in `$base/bin_external/prodigal/`
  - Link: `https://github.com/hyattpd/Prodigal`

4 How to continue and tips on running experiments!

After installation, it is best to first try running the toy example shown at the end of this document. It goes through all the steps from generating a target database to running StartLink. Since this is still in beta version, it will also help narrow down where issues might come from. Running commands with `-l INFO` should give enough information about what is going on. For debugging information, run with `-l DEBUG`.

The setup is still in its beta version. StartLink is currently limited to a particular directory structure, with a fixed type of input (i.e. full genome, with a specific database construction, see below). This was done to facilitate development and reduce potential for errors. I am currently working on refactoring and allowing a more diverse set of inputs, and relaxing the constraints on database structure so that users can use a BLASTp or Diamond BLASTp database that they already have. This is still very much work in progress.

Note: StartLink requires significant computational power and time. You can reduce this by tuning the parallelization parameters found in `$config/parallelization_defaults.conf`. It is recommended that you make a copy of this file and store it in `$config`. StartLink current supports PBS and multi-threading.

If you are interested in reproducing the results from scratch, then follow the document as is. If you are interested in testing out StartLink, then we recommend you start by running *H. salinarum* on the *Archaea* database. Start by downloading the data for genomes with verified start (see 6.1.1). If you already have the *Archaea* database downloaded, then move to 11. Otherwise, first construct the database using the instructions in 6.1.

5 Prebuilt databases and StartLink predictions

For convenience, we provide the StartLink(+) predictions and the DIAMOND databases used to generate them. These can be found at LINK

6 Downloading and installing

6.1 Data

Note: As previously mentioned the data used is on the order of hundreds of gigabytes. As such, if one is only interested in reproducibility, we provide pre-built databases (and even raw statistics from our existing runs).

We provide the databases for *Enterobacterales*, *Actinobacteria*, *Archaea*, and *FCB group*, and the sequence and label files for the genomes with verified starts: *E. coli*, *H. salinarum*, *N. pharaonis*, *M. tuberculosis*, and *R. denitrificans*. We also provide the steps to create a database with for any ancestor using data that can be downloaded from NCBI's website.

6.1.1 "Downloading" data for verified genomes

This data is already provided with this installation in `$data/../verified`. It should be copied or linked to the `$data` directory. The following command links it

```
awk -F "," '{if (NR > 1 && NF) print $1}' $lists/verified.list  | while read -r gcfid; do
  ln -sn $data/../verified/$gcfid $data/;
done
```

6.1.2 Downloading Assembly Summary File

```
$bin/download_assembly_summary_py.sh --database refseq --pf-output
    $metadata/refseq_assembly_summary.txt
```

6.1.3 Representative Genomes

```
# These lists are provided for reproducibility, since NCBI frequently updates them.
pf_refseq_arc=$lists/refseq_representative_archaea.list
pf_refseq_bac=$lists/refseq_representative_bacteria.list
pf_assembly_summary=$metadata/refseq_assembly_summary.txt

$bin/download_genomes_from_list_py.sh --pf-genome-list $pf_refseq_arc
    --pf-assembly-summary $pf_assembly_summary -l INFO
$bin/download_genomes_from_list_py.sh --pf-genome-list $pf_refseq_bac
    --pf-assembly-summary $pf_assembly_summary -l INFO
```

6.1.4 Constructing Taxonomy Tree

To easily download genomes by clade, we first construct a taxonomy tree form NCBI's taxonomy information.

```
pd_work=$tmp/tree

mkdir -p $pd_work

$bin/download_taxonomy_dump_py.sh --pd-output $metadata/taxdump
$bin/build_taxonomy_tree_py.sh --pf-nodes-dmp $metadata/taxdump/nodes.dmp --pf-names-dmp
    $metadata/taxdump/names.dmp --pf-tree $pd_work/tree.pkl
```

6.1.5 Download sequence/label files for different clades, and create Blast databases

Now, we can download genomes from different clades. In the paper, 5 clades are referenced (4 of which are used for the large scale analysis). If you want to change which clade is downloaded, simply change the array `clades` in the below script. The names are defined by NCBI's `name_txt` variable in it's assembly summaries.

```
function mk_path_friendly() {
  echo "$1" | tr " " "_" | tr [:upper:] [:lower:];
}

dbt=refseq     # database type
pf_tree=$tmp/tree/tree.pkl
pf_ass_sum_comb=$metadata/${dbt}_assembly_summary.txt

# Modify this array based on which genome data you want (e.g. set as "Archaea" if you are
    only interested
# in archaeal genomes, or add any valid name_txt id from the taxonomy tree).
# Warning: This downloads all genomes under that clade (one per taxonomy ID)
declare -a clades=("Enterobacterales" "Actinobacteria" "Alphaproteobacteria" "FCB group"
    "Archaea")

# loop over clades; download data under each clade
for cl in ${clades[@]}; do
  dn_cl=$(mk_path_friendly "$cl")
  $bin/download_genomes_for_clade_py.sh --pf-tree $pf_tree --pf-assembly-summary
      $pf_ass_sum_comb --clade-id $cl --clade-id-type "name_txt"
      --favor-assembly-level-order --genomes-per-taxid 1 --pf-output-list
      $lists/${dbt}_${dn_cl}.list
done
```

Construct Diamond Blastp databases

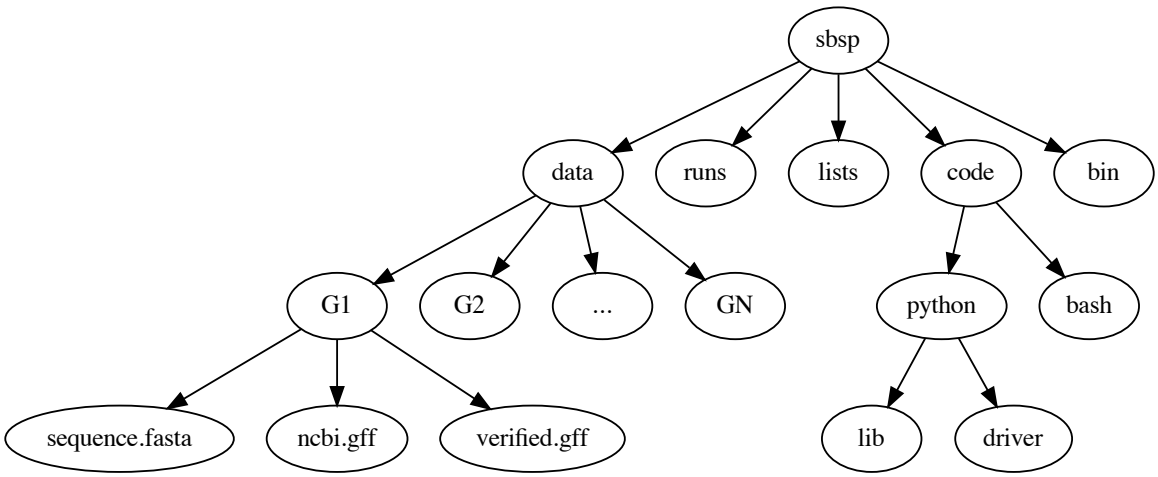
```
for cl in "${clades[@]}"; do
  dn_cl=$(mk_path_friendly "$cl")

  pd_work="$tmp/extract_sequences/${dn_cl}"
  mkdir -p $pd_work

  cd $pd_work

  pf_list=$lists/${dbt}_${dn_cl}.list
  pf_faa=$pd_work/${dbt}_${dn_cl}.faa
  pf_db=$db/${dbt}_${dn_cl}.dmnd

  # extract sequences
  $bin/extract_annotated_sequences_py.sh --pf-genome-list $pf_list --pf-output $pf_faa

  # build blast
  $bin/build_blast_db_py.sh --pf-sequences $pf_faa --pf-db $pf_db

  # clean up sequence file
  [[ -f $pf_faa ]] & rm $pf_faa
done

cd $base
```

6.1.6 Download query genomes from list

```
pf_query_large=$lists/selected_query.list
pf_ass_sum_query_large=$metadata/assembly_summary_query_large.txt
$bin/download_genomes_from_list_py.sh --pf-genome-list $pf_query_large
    --pf-assembly-summary $pf_ass_sum_query_large  -l INFO
```

7 Code and data structure

After installing StartLink, you will have the following structure:

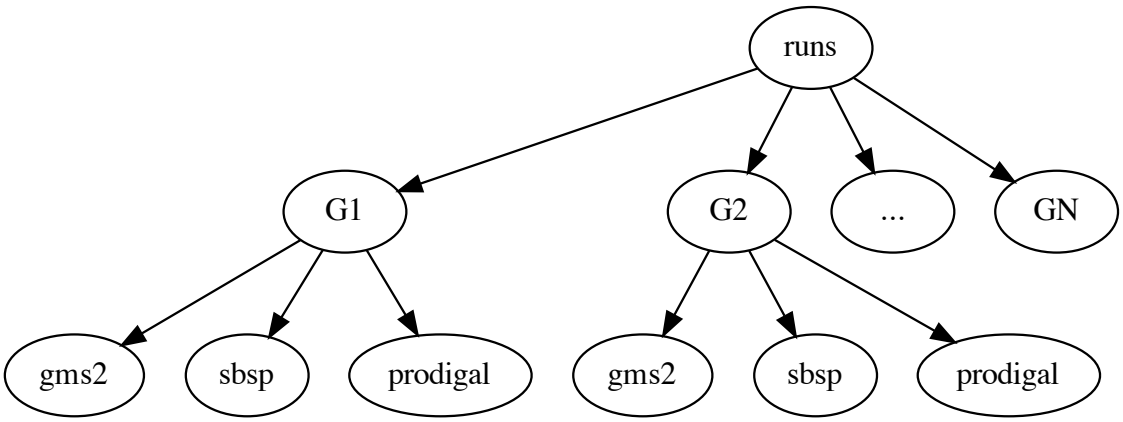


The `bin` directory contains all executables related to StartLink, while the `bin_external` may contain external tools, such as GeneMarkS-2 or Prodigal.

The `data` directory will contain raw genome files (sequence and annotation labels) downloaded from NCBI. In particular, upon initial download of the code, it should contain the genomic sequences for the genomes with experimentally verified gene-starts.

The `list` directory has files that contain different lists of genomes (for example, those with verified genes, those selected as NCBI query genomes, etc. . .)

Finally the `runs` directory will contain runs of different tools, such as StartLink, GeneMarkS-2, or Prodigal (as well as one for NCBI's `PGAP`). These will be placed in a subdirectory per genome, as shown below.



8 Setting up

Since much of the analysis is done by comparing StartLink to NCBI's PGAP, GeneMarkS-2, and/or Prodigal, we first need to run these tools and add the results to the run directory. The following script is capable of doing that (note, depending on which analysis you want to reproduce, you may not need to run the tools on all lists):

```
# Enable if you have PBS installed
pbs_options=""
#pbs_options="--pf-parallelization-options $config/parallelization_pbs_1.conf"

function run_tools_on_archaea() {
  pf_list="$1"

  $bin/run_tool_on_genome_list_py.sh --tool gms2 --pf-genome-list $pf_list --type archaea
      -l INFO --pd-work $runs ${pbs_options}
  $bin/run_tool_on_genome_list_py.sh --tool prodigal --pf-genome-list $pf_list --type
      archaea -l INFO --pd-work $runs ${pbs_options}
}

function run_tools_on_bacteria() {
  pf_list="$1"

  $bin/run_tool_on_genome_list_py.sh --tool gms2 --pf-genome-list $pf_list --type
      bacteria -l INFO --pd-work $runs ${pbs_options}
```

```
  $bin/run_tool_on_genome_list_py.sh --tool prodigal --pf-genome-list $pf_list --type
      bacteria -l INFO --pd-work $runs ${pbs_options}
}

pf_rep_arc=$lists/refseq_representative_archaea.list
pf_rep_bac=$lists/refseq_representative_bacteria.list

pf_list_verified_arc=$lists/verified_archaea.list
pf_list_verified_bac=$lists/verified_bacteria.list

pf_list_qncbi_arc=$lists/qncbi_archaea.list
pf_list_qncbi_bac=$lists/qncbi_bacteria.list


# Representative genomes
run_tools_on_archaea $pf_rep_arc
run_tools_on_bacteria $pf_rep_bac

# Verified genomes
run_tools_on_archaea $pf_list_verified_arc
run_tools_on_bacteria $pf_list_verified_bac

# NCBI query genomes
run_tools_on_archaea $pf_list_qncbi_arc
run_tools_on_bacteria $pf_list_qncbi_bac
```

9 Experiments

Unless otherwise noted, these variables (when applicable) will have the following values

```
pf_list_verified=$lists/verified.list  # verified genomes
pf_list_qncbi=$lists/qncbi.list   # query genomes

# database and configuration files
pf_db_index=$db/index.csv  # database location files
pf_sbsp_options=$config/sbsp_defaults.conf # sbsp config file
pf_pbs_options=$config/parallelization_defaults.conf   # PBS config file

# PBS options
toggle_pbs="--pf-parallelization-options $pf_pbs_options"  # if PBS not installed, set
    this option to empty: ""
sg=8   # number of genomes to run simutaneously (low number recommended)
```

9.1 Difference in 5' predictions on Representative Genomes

9.1.1 Data download

```
pf_rep_bac=$lists/refseq_representative_bacteria.list
pf_rep_arc=$lists/refseq_representative_archaea.list
pf_assembly_bac=$metadata/assembly_summary.txt
# $bin/download_from_ncbi_py.sh --pf-assembly-summary $pf_assembly_bac --pf-data $data
    --pf-output-list
```

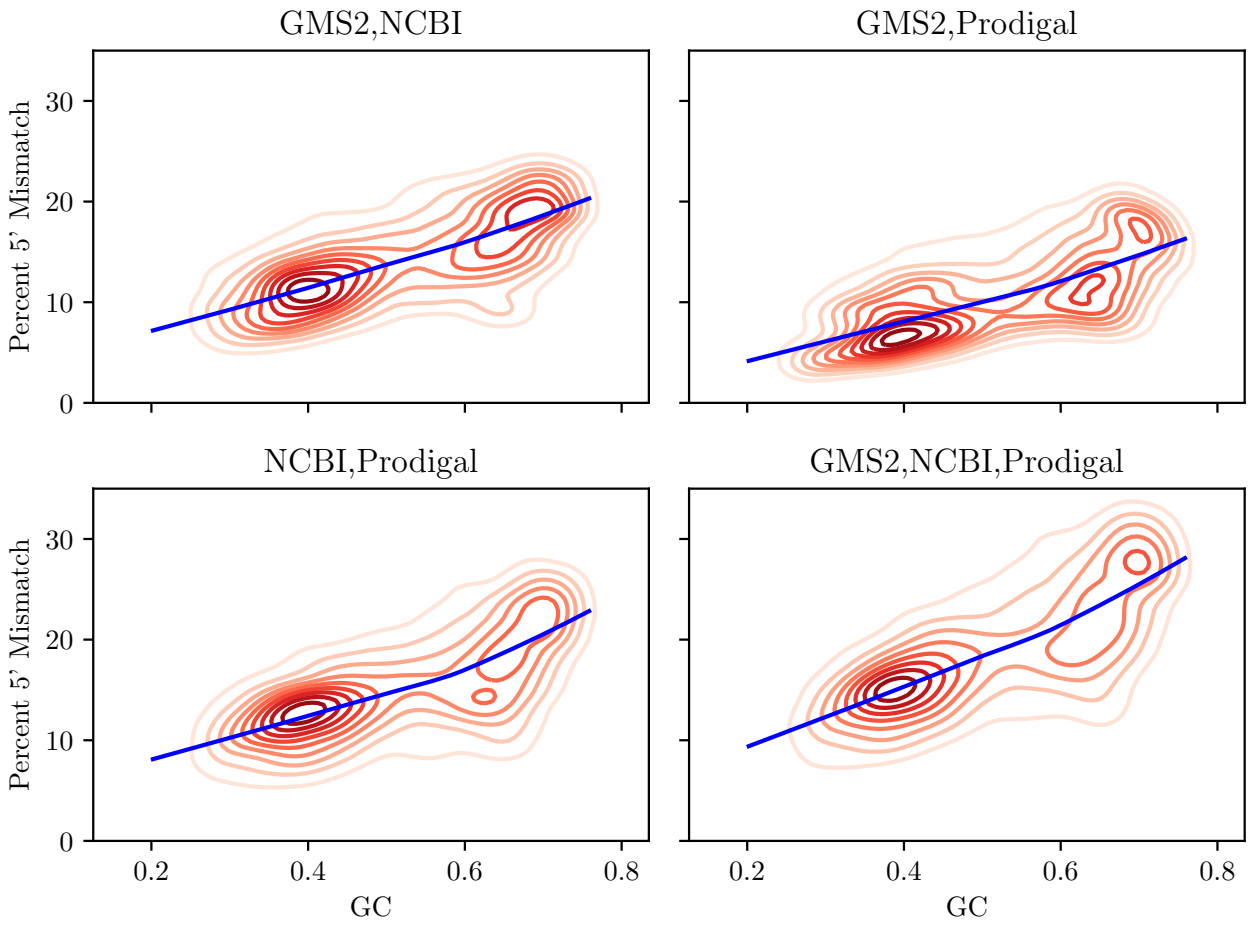
```
# link ncbi as "tool" (for easy comparison wwith other tools)
cat $pf_rep_bac $pf_rep_arc | grep -v gcfid | cut -f1 -d, | while read -r line; do
  mkdir -p $runs/$line; mkdir -p $runs/$line/ncbi;
  ln -s $data/$line/ncbi.gff $runs/$line/ncbi/ncbi.gff ;
done
```

9.1.2 Collect statistics

We can now collect the statistics and create the figures to compare GMS2, Prodigal, and NCBI predictions.

```
pd_work=$tmp/stats_refseq_vs_tools
mkdir -p $pd_work
cd $pd_work

pf_stats=$pd_work/stats_tools.csv

$bin/stats_tools_5prime_py.sh --pf-genome-lists $pf_rep_bac $pf_rep_arc --list-names
    Bacteria Archaea --dn-tools gms2 prodigal ncbi --tool-names GMS2 Prodigal NCBI
    --pf-output $pf_stats

# create figures
$bin/viz_stats_tools_5prime_py.sh --pf-stats $pf_stats

cd $base
```

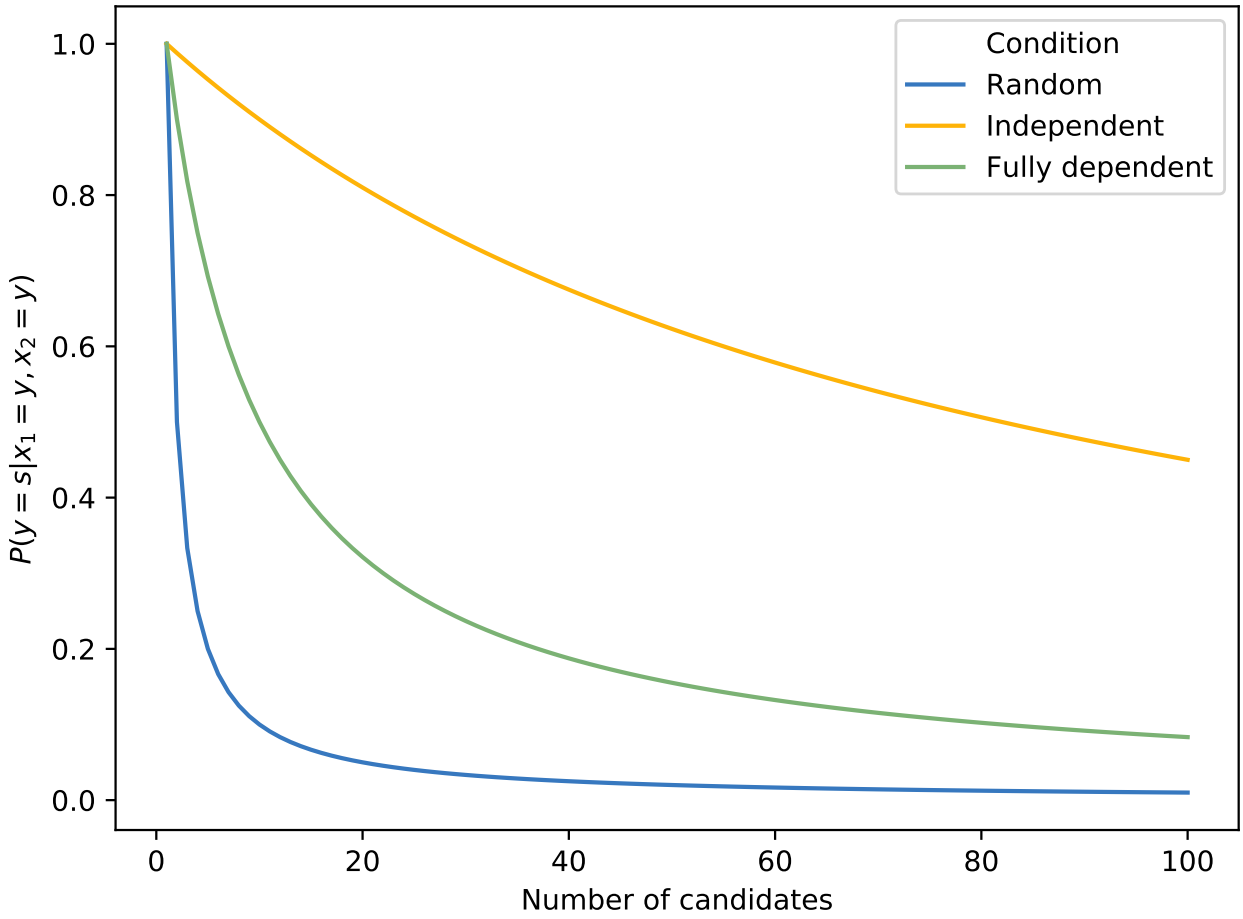
This should now create a file containing the following image



9.2 Theoretical view of Independence

While not technically an experimental result, we provide the code to generate this graph for convenience. The sensitivity of the non-random algorithms A_1 and A_2 are set to 0.9, but the user can easily change them (from within) to observe the change in behavior. What remains constant is the improvement of independent algorithms over fully dependent (and random) algorithms..

```
$bin/independent_predictions_py.sh
```



9.3 Genomes with genes with verified starts

9.3.1 Running StartLink

```
mkdir $tmp/verified
cd $tmp/verified

# run SBSP
$bin/sbsp_on_genome_list_py.sh --pf-q-list $pf_list_verified --simultaneous-genomes $sg
    --pd-work $runs --pf-sbsp-options $pf_sbsp_options  --pf-db-index $pf_db_index
    $toggle_pbs

cd $base
```

9.3.2 Collecting statistics

```
mkdir $tmp/verified_stats
cd $tmp/verified_stats

# collect statistics per query gene (comparing StartLink, GMS2, and verified genes)
$bin/stats_per_query_gene_py.sh --pf-genome-list $pf_list_verified --pf-output-summary
    summary.csv --verified

cd $base
```

9.3.3 Visualizing

```
cd $tmp/verified_stats

$bin/viz_stats_genome_level_py.sh --pf-data summary.csv

cd $base
```

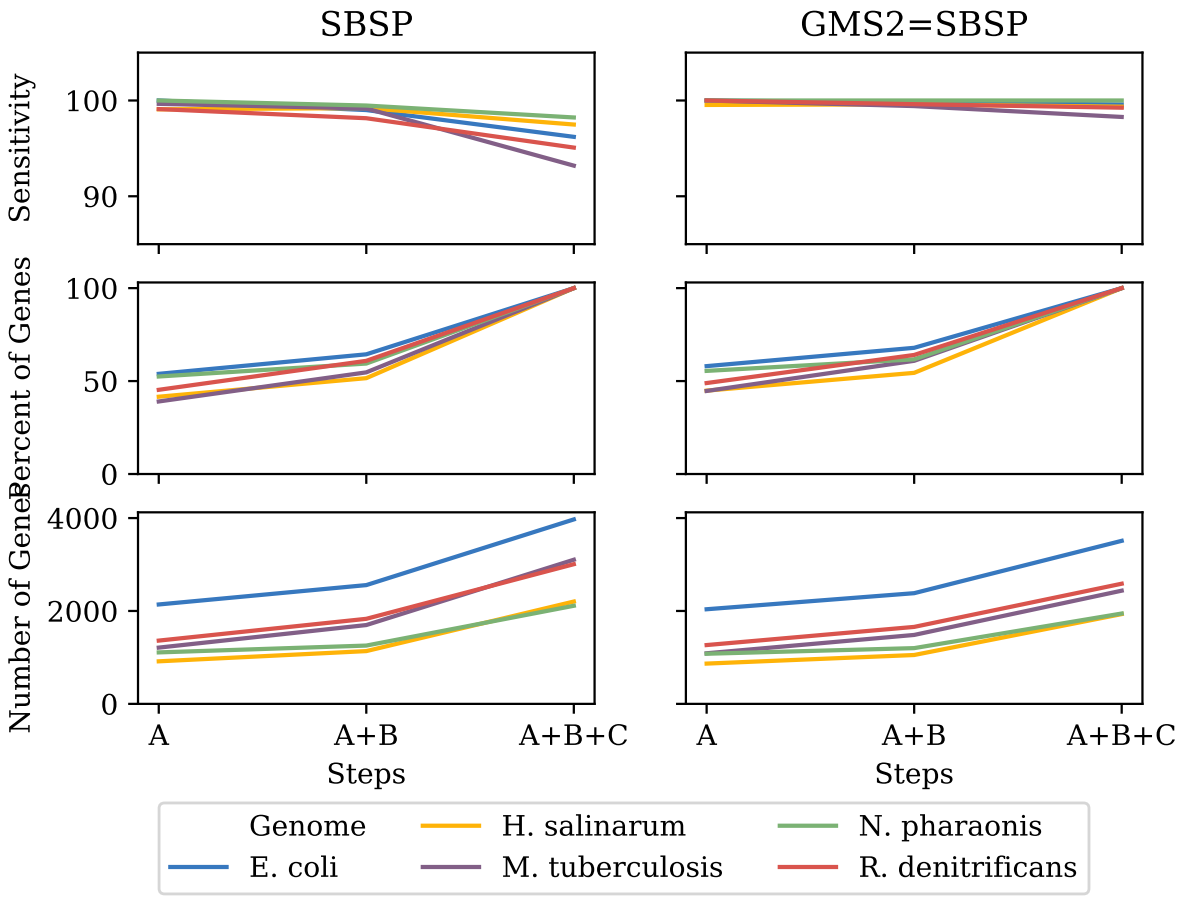
This will produce two files, `error.csv` and `coverage.csv` containing the following two tables.
Error

Genome	Verified	SBSP	GMS2	GMS2=SBSP
E. coli	769	96.204188	97.001304	99.582754
H. salinarum	530	97.489540	98.679245	99.354839
M. tuberculosis	701	93.197279	90.401146	98.282443
N. pharaonis	315	98.226950	99.047619	100.000000
R. denitrificans	526	95.081967	96.571429	99.248120

Coverage

Genome	Verified	SBSP	GMS2	GMS2=SBSP
E. coli	769	99.349805	99.739922	93.498049
H. salinarum	530	90.188679	100.000000	87.735849
M. tuberculosis	701	83.880171	99.572040	74.750357
N. pharaonis	315	89.523810	100.000000	87.301587
R. denitrificans	526	81.178707	99.809886	75.855513

It also produces the per-step analysis on the verified set of genes.



9.4 Larger set of query genomes

9.4.1 Running StartLink

Prewarning, running this analysis can take a long time. Our estimate is roughly 5 days on 20 compute nodes with 8 processors each, though that number can vary based on how databases are setup, where they are located, and the cost of accessing them (e.g. databases can be copied to each node beforehand, making access much cheaper and prevent bottlenecks).

In that respect, we have also provided a CSV file containing the per-query analysis of all genes in this set, which is used for visualization of results.

```
mkdir $tmp/large
cd $tmp/large

# run SBSP
$bin/sbsp_on_genome_list_py.sh --pf-q-list $pf_list_qncbi --simultaneous-genomes $sg
    --pd-work $runs --pf-sbsp-options $pf_sbsp_options  --pf-db-index $pf_db_index
    $toggle_pbs

cd $base
```

9.4.2 Collecting statistics

```
mkdir $tmp/large_stats
cd $tmp/large_stats

# collect statistics per query gene (comparing SBSP, GMS2, and verified genes)
$bin/stats_per_query_gene_py.sh --pf-genome-list $pf_list_qncbi --pf-output-summary
    summary.csv

cd $base
```

9.4.3 Visualizing

All images regarding the large-scale comparisons can be generated via a single script. Note that the contour plots are computationally expensive and may take ~1 hour to generate. Therefore, they are turned off by default. To enable them, run the command with the option `--with-contours`.

```
cd $tmp/large_stats

$bin/viz_stats_clade_level_py.sh --pf-data summary.csv

cd $base
```

(NCBI, GMS2=SBSP)$) based on the minimum and maximum Kimura distances between a query and its targets. The color bar measures the sensitivity rate, with brighter colors indicating higher sensitivity

10 Toy Example

This section lists a set of commands to generate a toy database and and run H. salinarum predictions on it. This is just meant for the user to test out whether the system has been installed correctly.

Make sure you have GeneMarkS-2 installed under `$base/bin_external/gms2/`, and `Diamond` and `ClustalO` should be installed and in your path.

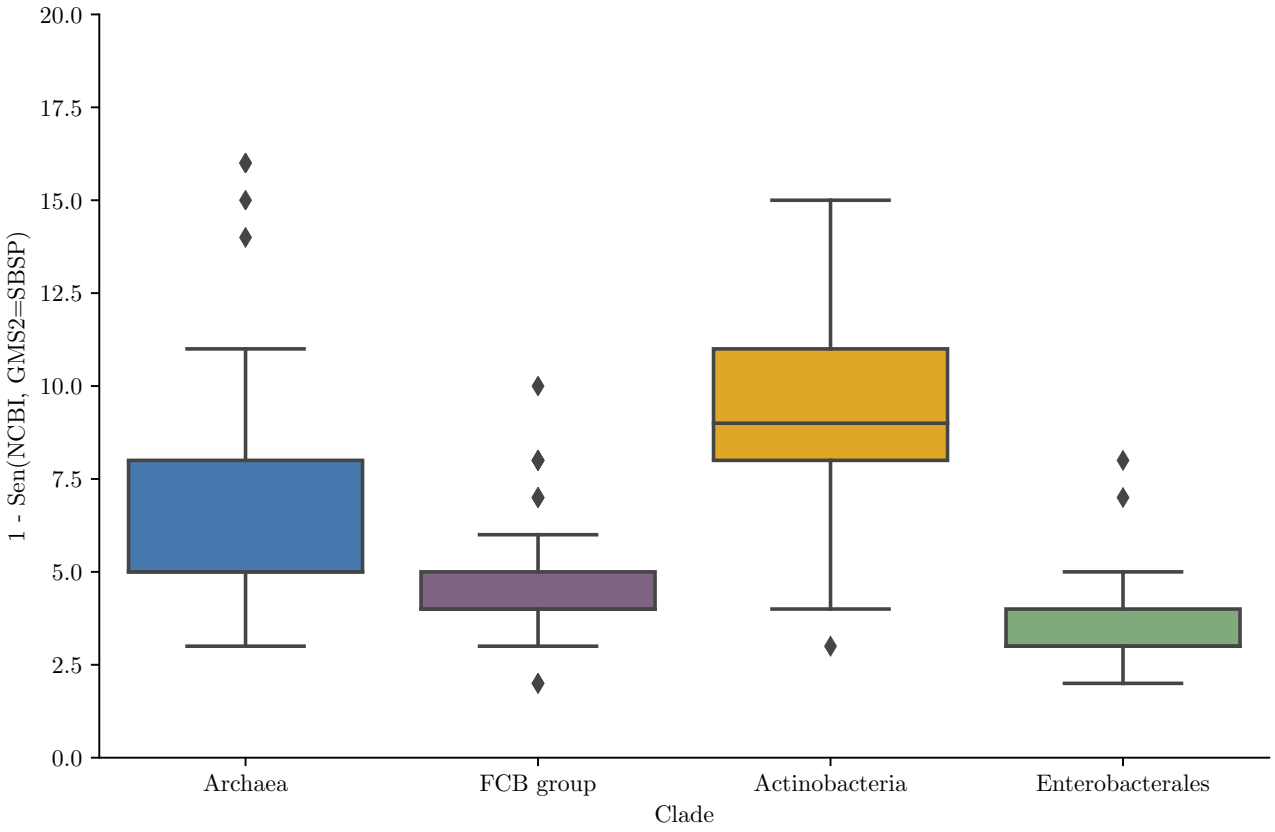


Figure 1: The 5' error rate of NCBI compared to GMS2=SBSP for query genomes in different clades

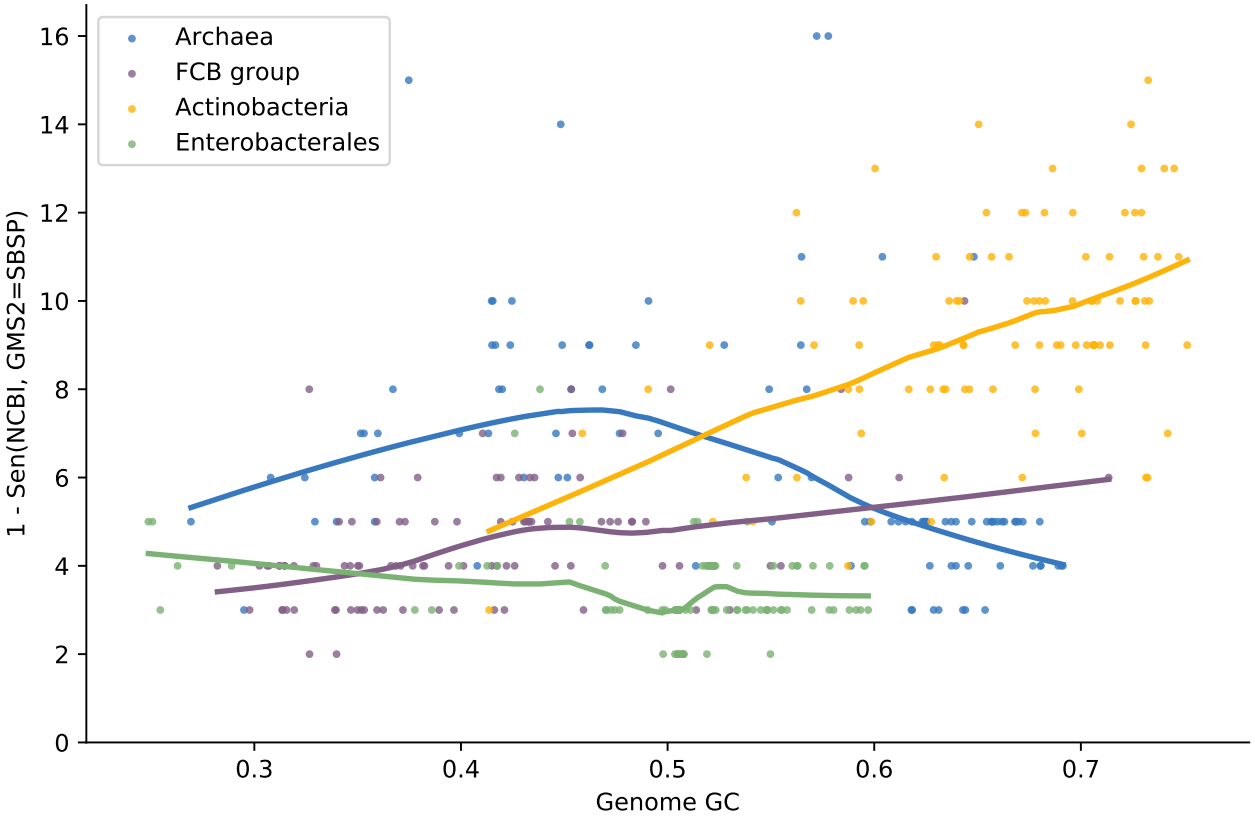


Figure 2: The 5' error rate of NCBI compared to GMS2=SBSP, as a function of genome GC

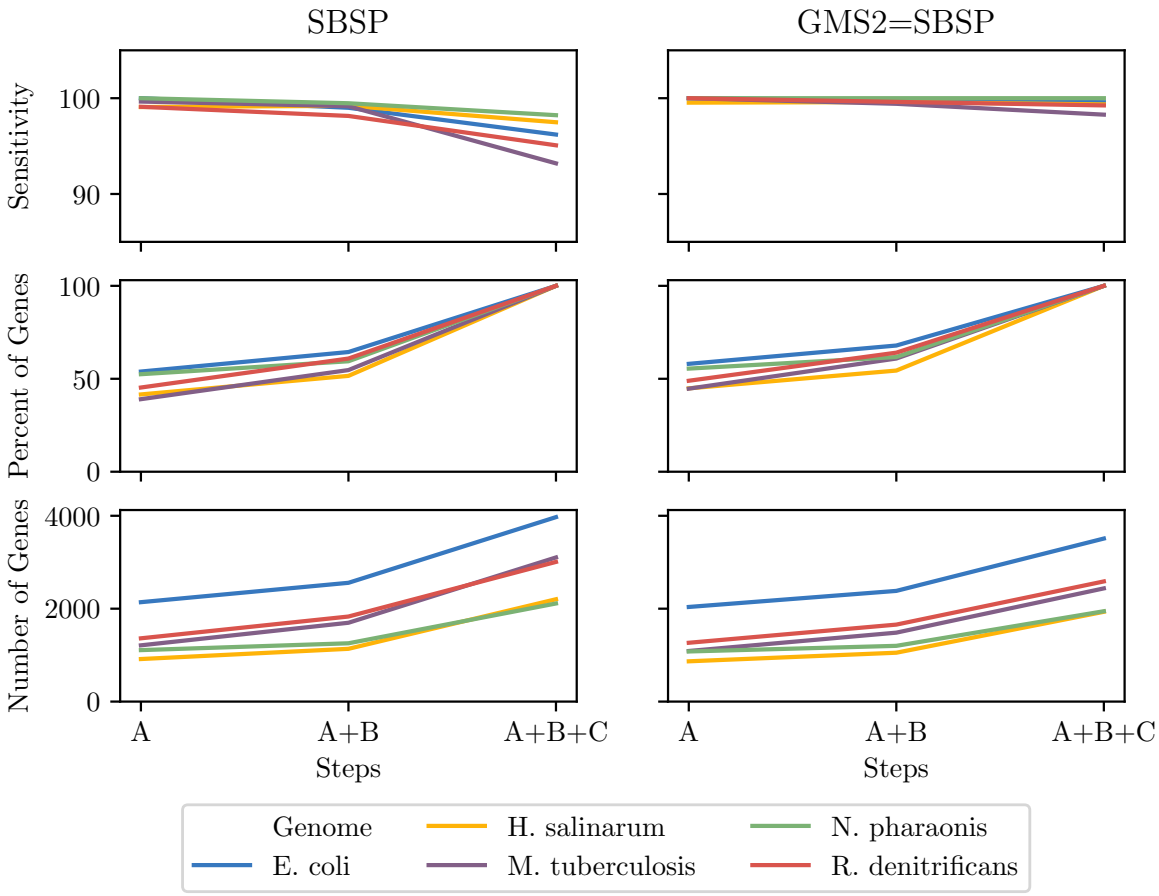


Figure 3: Left: The sensitivity for each SBSP step on the set of verified genes (top), and the percentage (middle) and number (bottom) of SBSP genes predicted by step A alone, steps A and B, and all steps together. Right: Same analysis, for GMS2=SBSP.

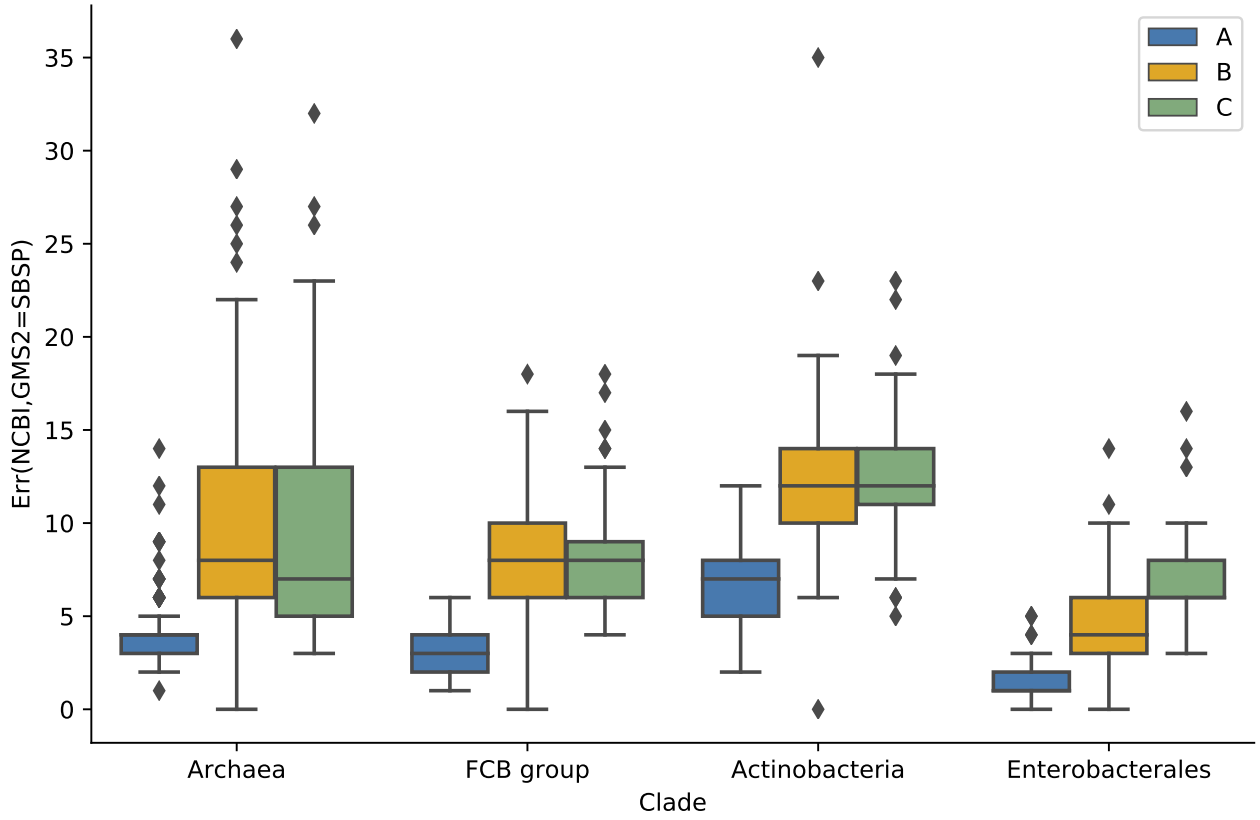


Figure 4: The 5' error rate of NCBI compared to GMS2=SBSP, shown per step of SBSP

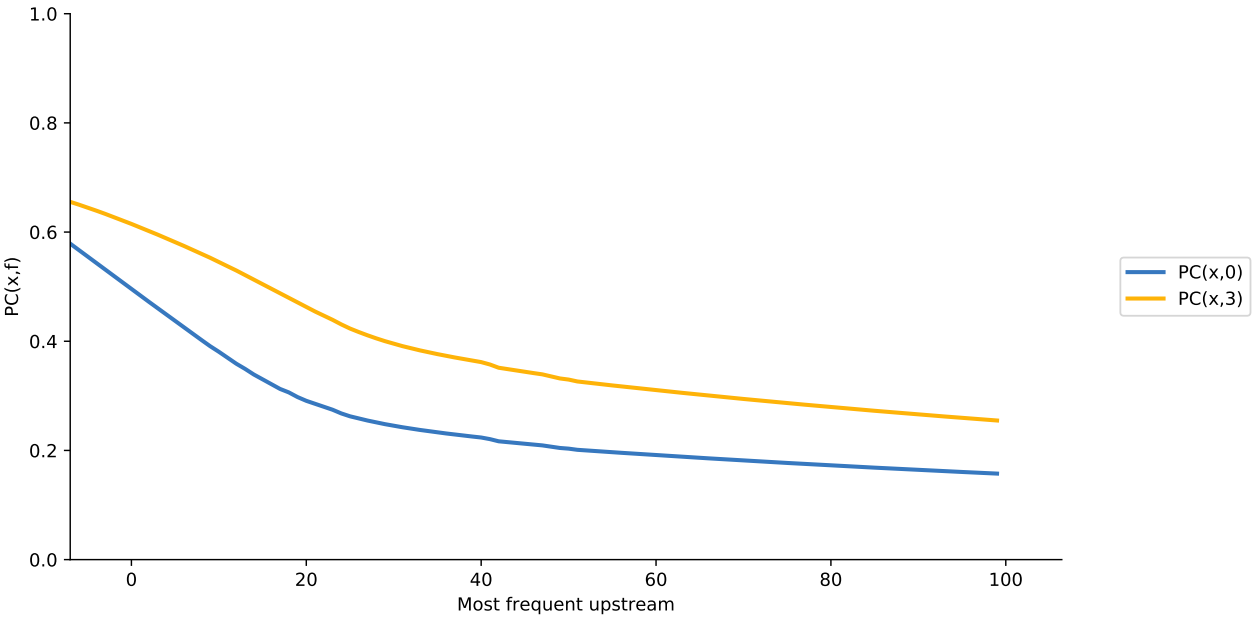


Figure 5: The variation in proximity consistency as the distance to the upstream gene increases

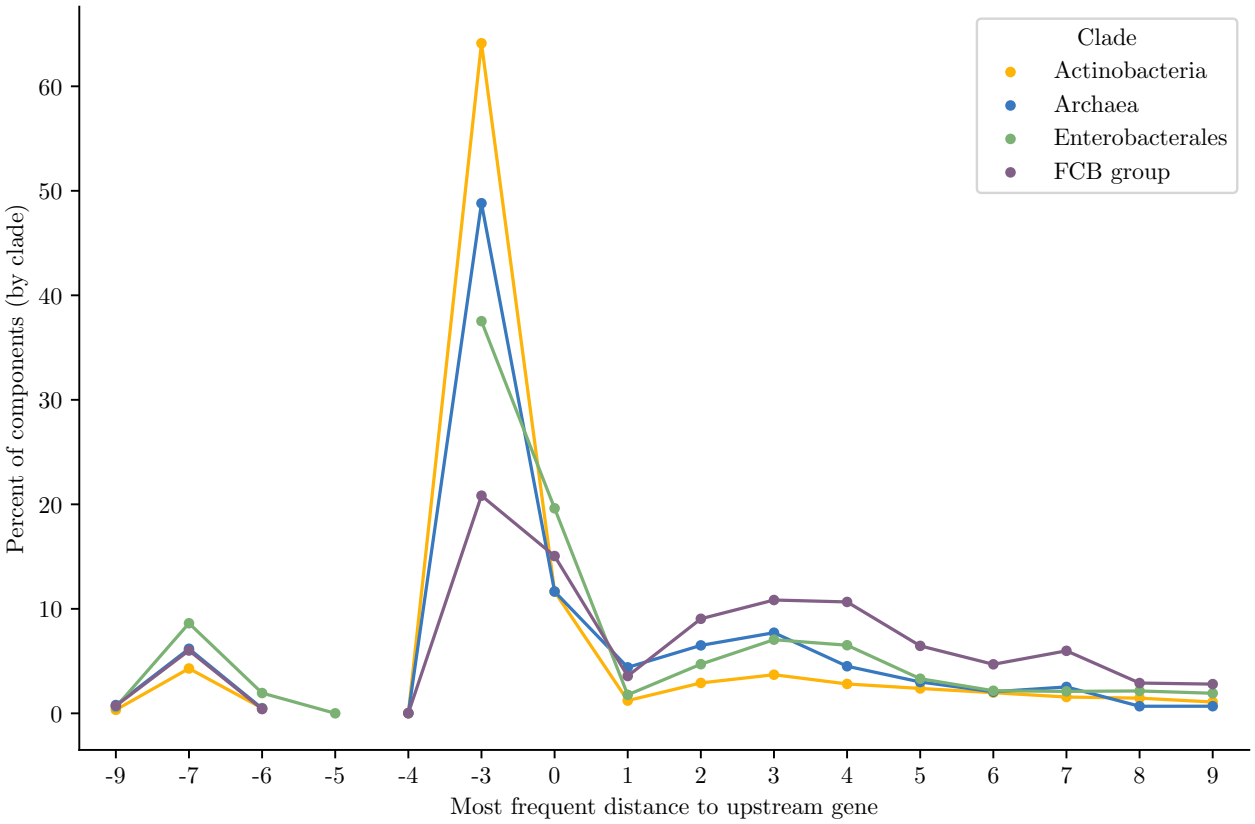


Figure 6: The percentages of components whose most frequent upstream distance lies within the -10 and +10 *nt* range. A component is defined as a single query and its targets

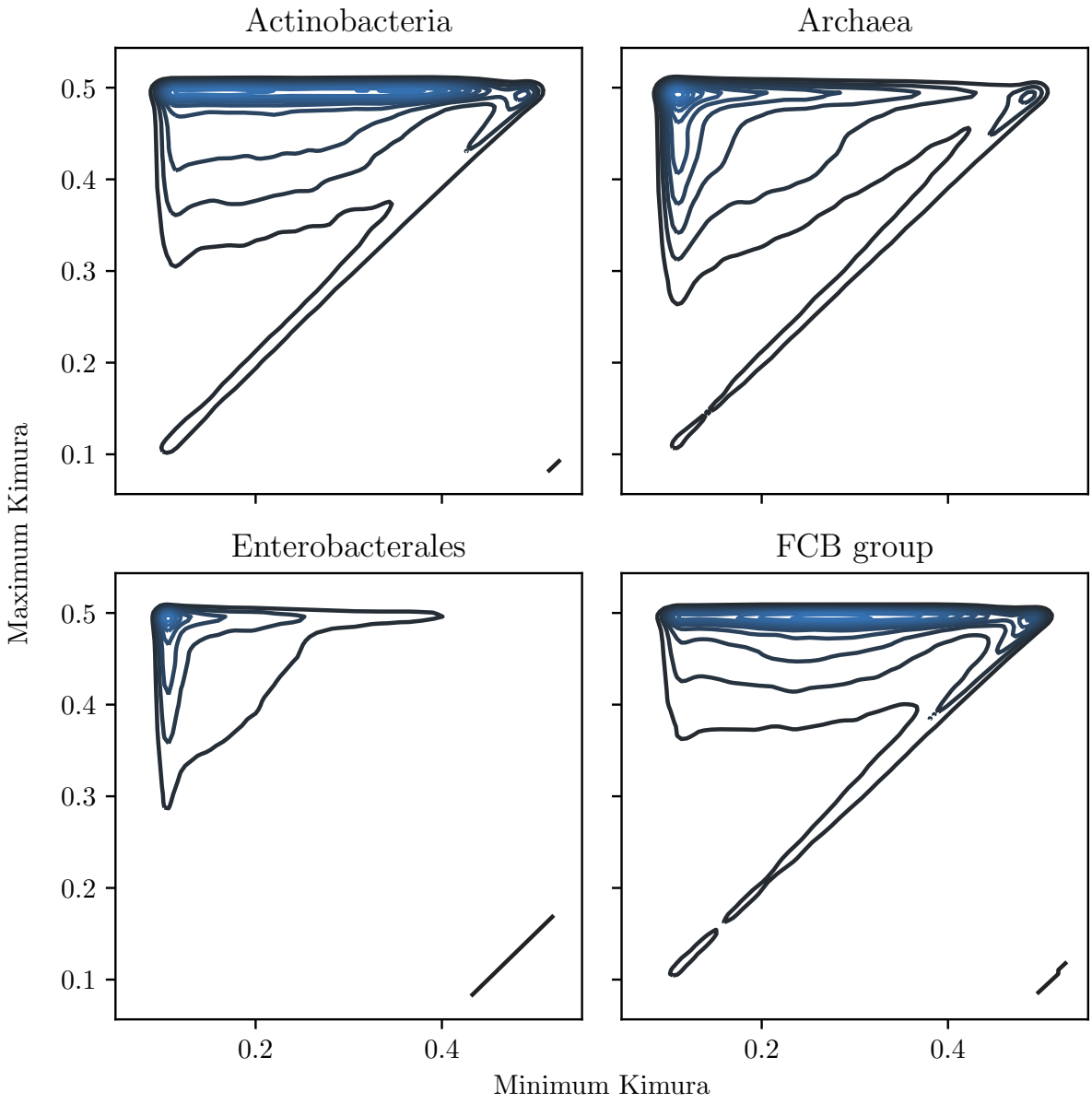


Figure 7: The distribution of queries by minimum and maximum Kimura distance to their orthologs. This shows that most query genes in *Enterobacterales* will find an orthologs that spread the range from 0.1 to 0.5 Kimura, whereas many in *Actinobacteria* have a minimum Kimura distance of above 0.3 and even 0.4

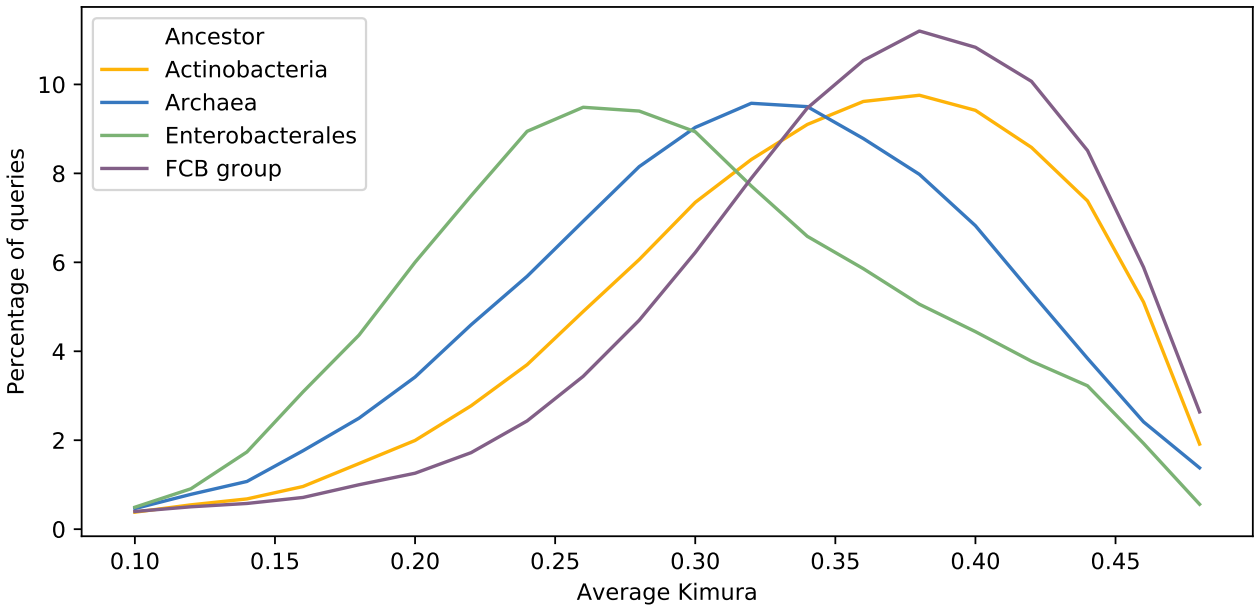


Figure 8: The distribution of average Kimura distances (per component). The y-axis shows the percentage of queries (and thus, components) that have a particular average Kimura distance to its orthologs

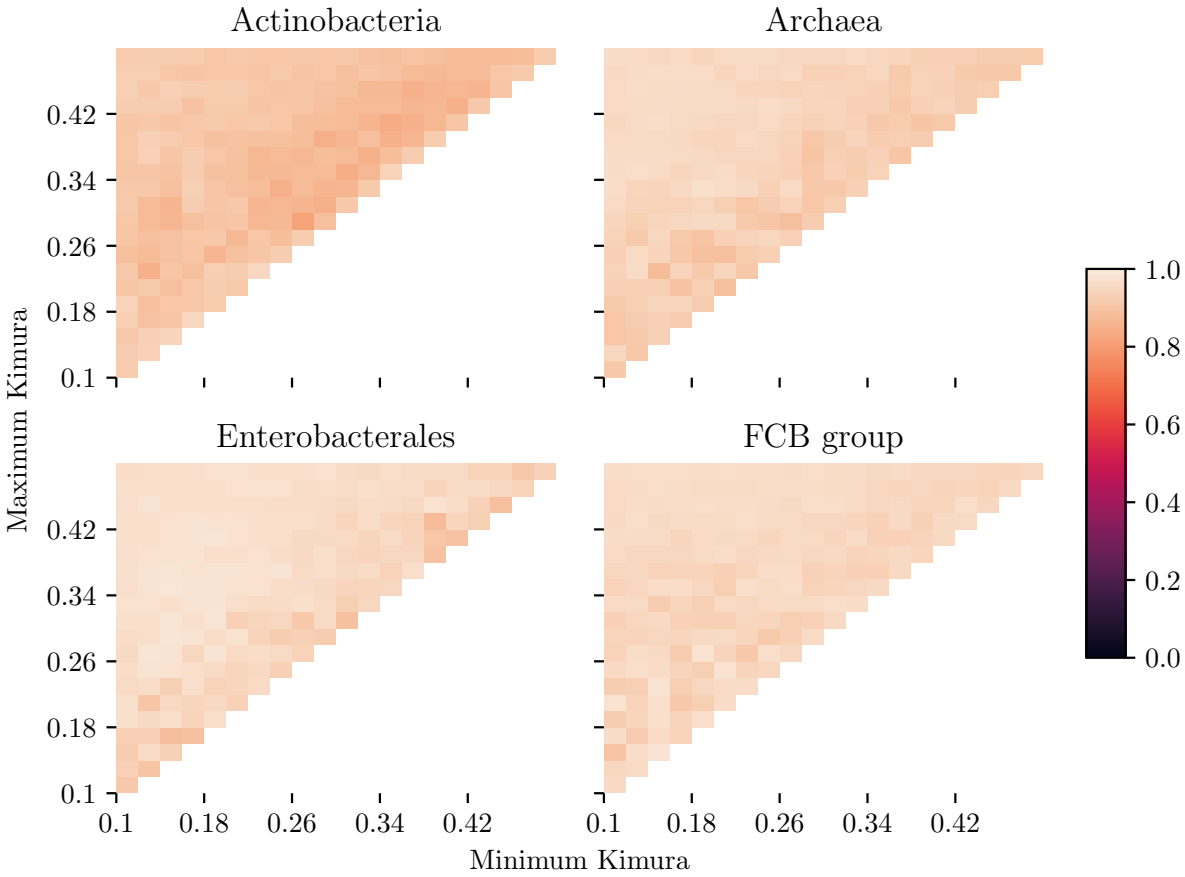


Figure 9: The 5' sensitivity rate of NCBI compared to GMS2=SBSP (i.e. $Acc

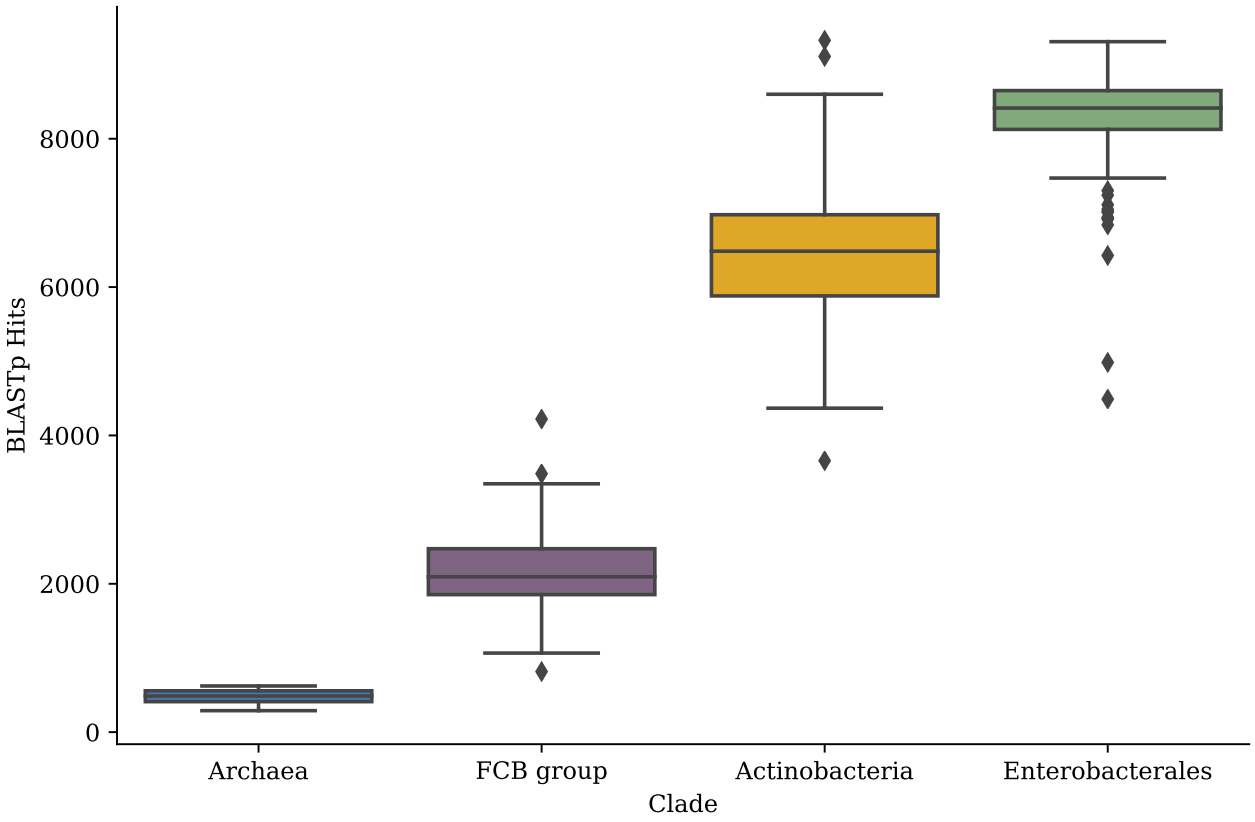


Figure 10: Distribution of raw blast hits across clades for the set of query genomes in Table~**??**. Left: The raw number of BLAST hits per clade. Right: The cumulative percentage of queries with *at most* N BLASTp hits, where N varies from 0 to 5,000. The shaded band shows the standard deviation (per clade) across query genomes

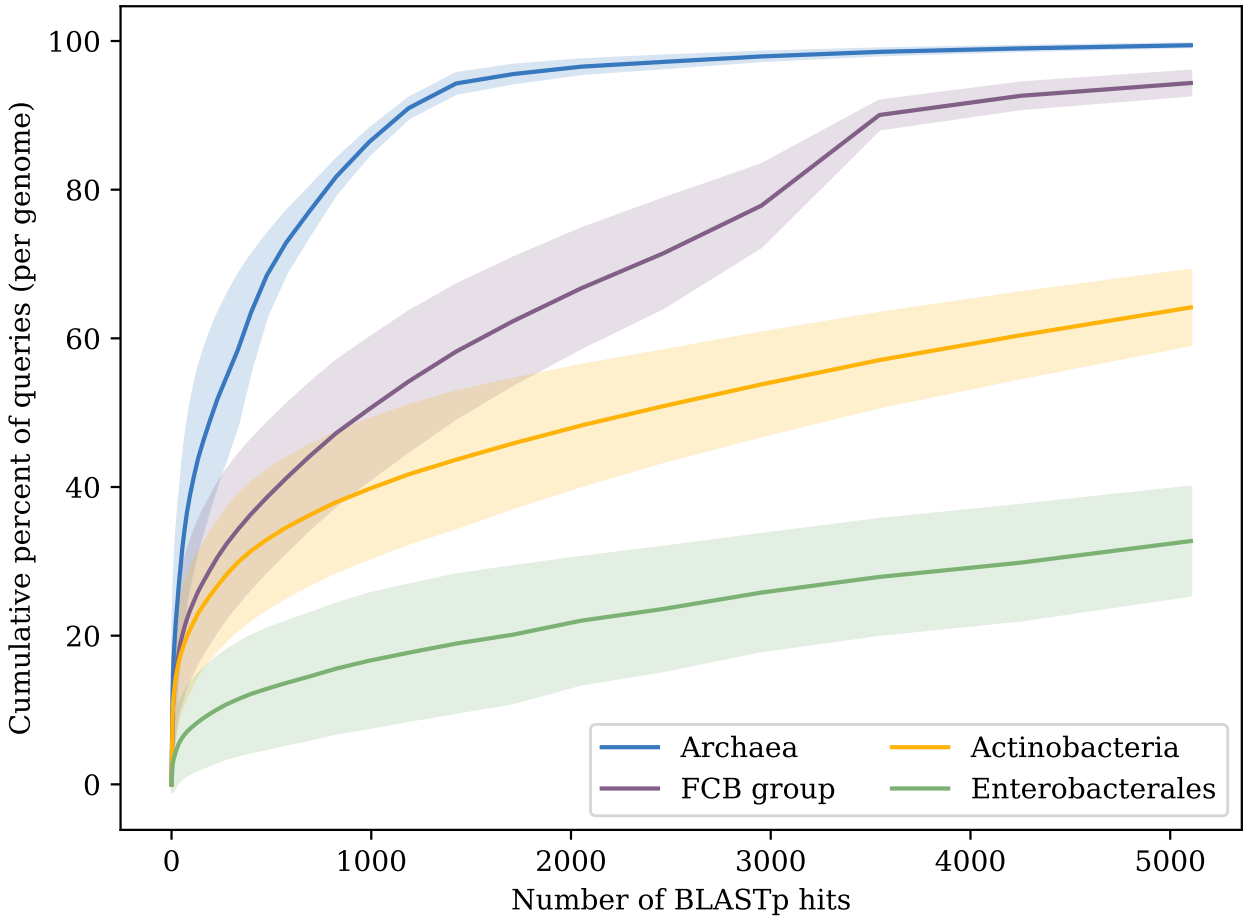


Figure 11: Distribution of raw blast hits across clades for the set of query genomes in Table~**??**. Left: The raw number of BLAST hits per clade. Right: The cumulative percentage of queries with *at most* N BLASTp hits, where N varies from 0 to 5,000. The shaded band shows the standard deviation (per clade) across query genomes

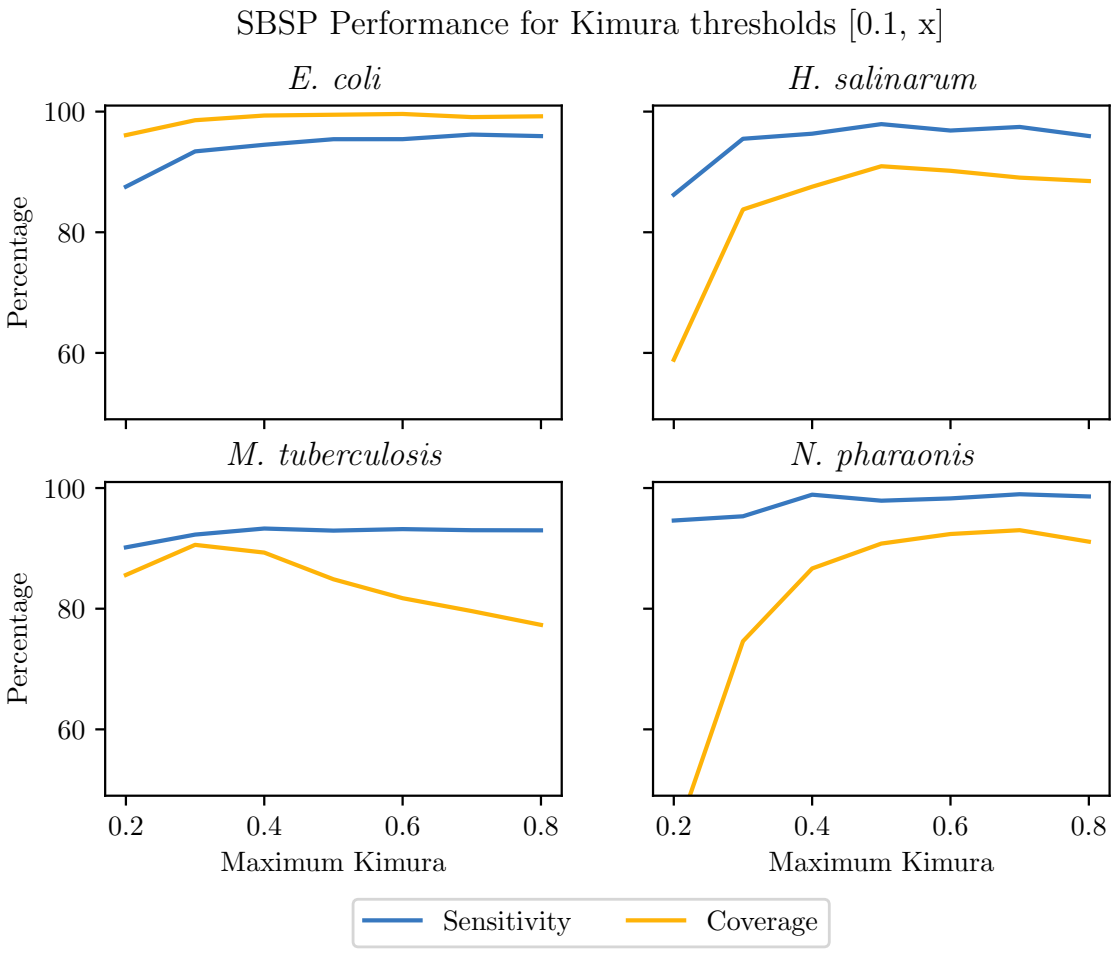


Figure 12: The effect of changing the maximum Kimura threshold on SBSP's sensitivity and coverage rates. The minimum Kimura threshold is fixed to 0.1, and $x \in \{0.2, 0.3, ..., 0.8\}$

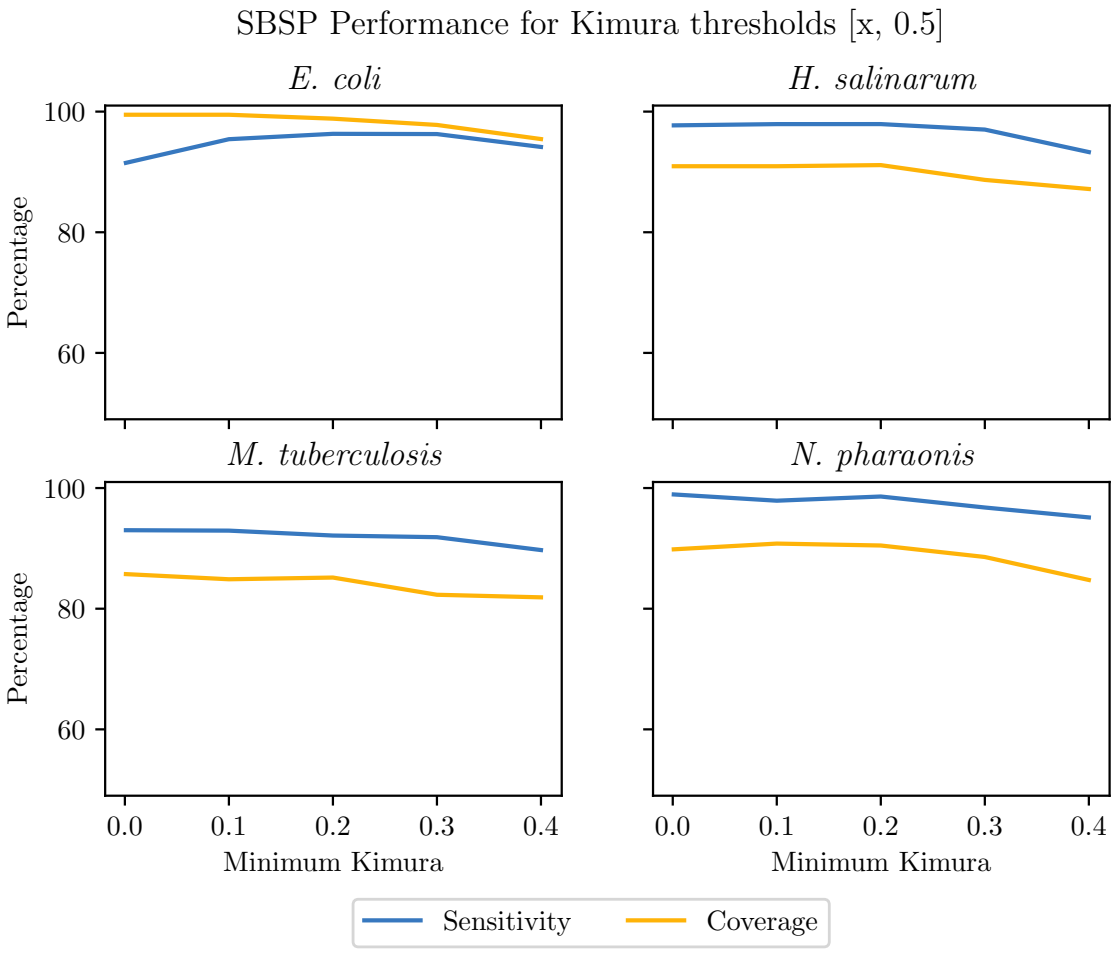


Figure 13: The effect of changing the minimum Kimura threshold on SBSP's sensitivity and coverage rates. The maximum Kimura threshold is fixed to 0.5, and $x \in \{0.001, 0.1, 0.2, 0.3, 0.4\}$

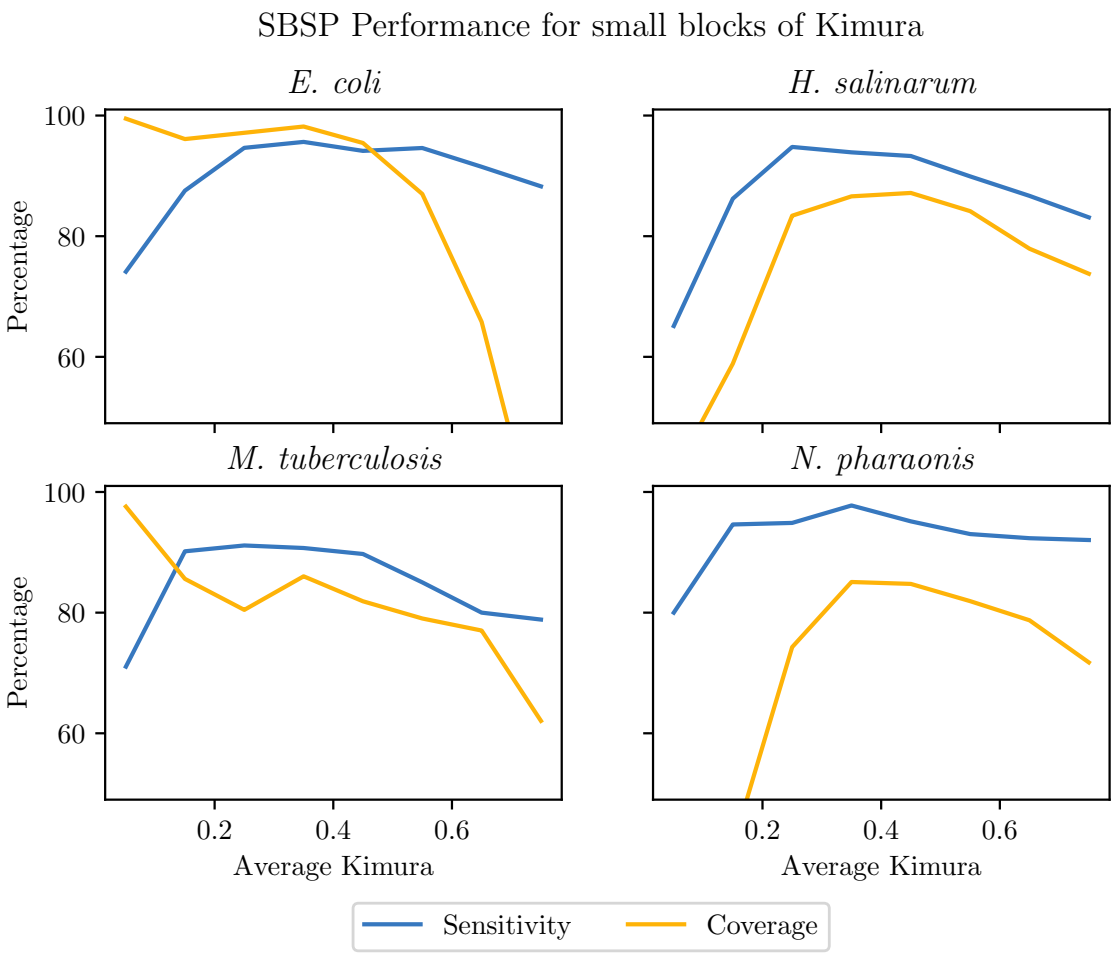


Figure 14: The performance of SBSP on small intervals of Kimura ranges: $[0.001, 0.1], [0.1, 0.2], [0.2, 0.3] \ldots [0.7, 0.8]$. The x-axis shows the mean Kimura of a block; e.g., for range $[a, b]$, the average is $(b + a)/2$

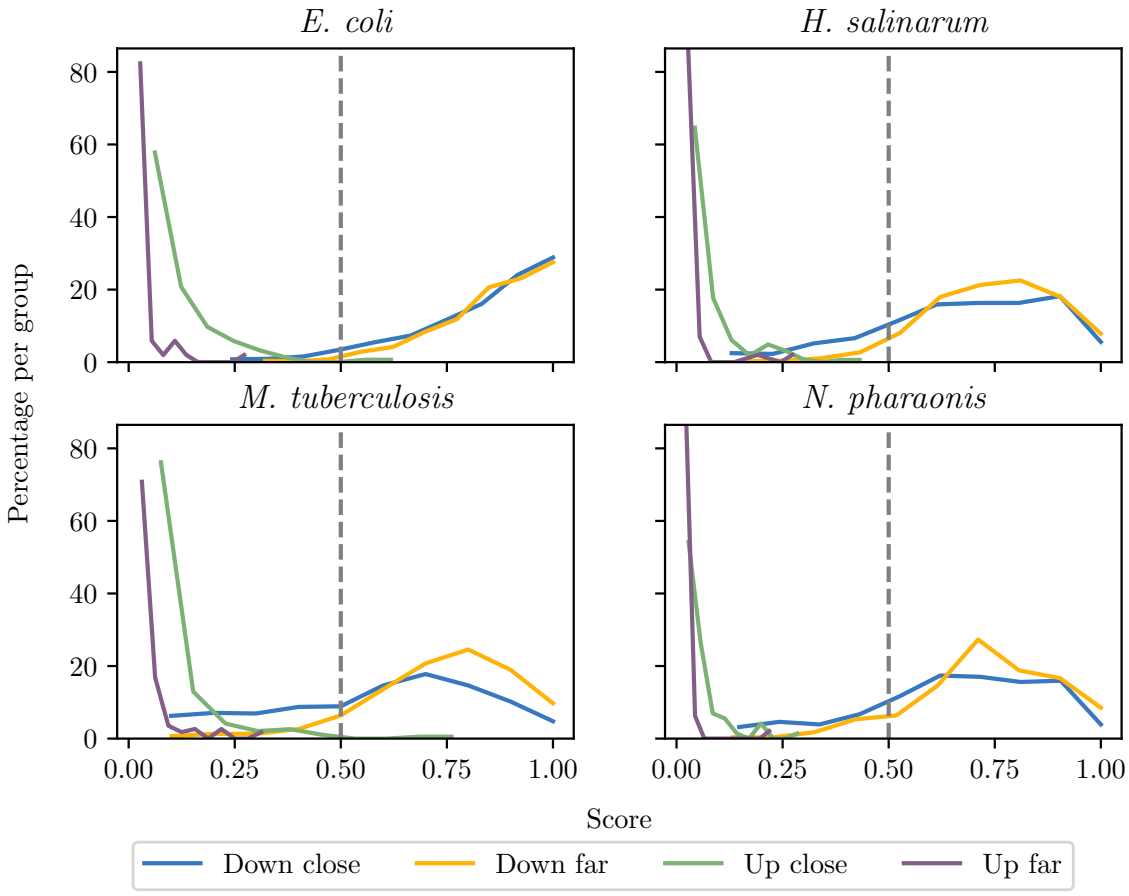


Figure 15: Distribution of block conservation scores in regions around verified starts

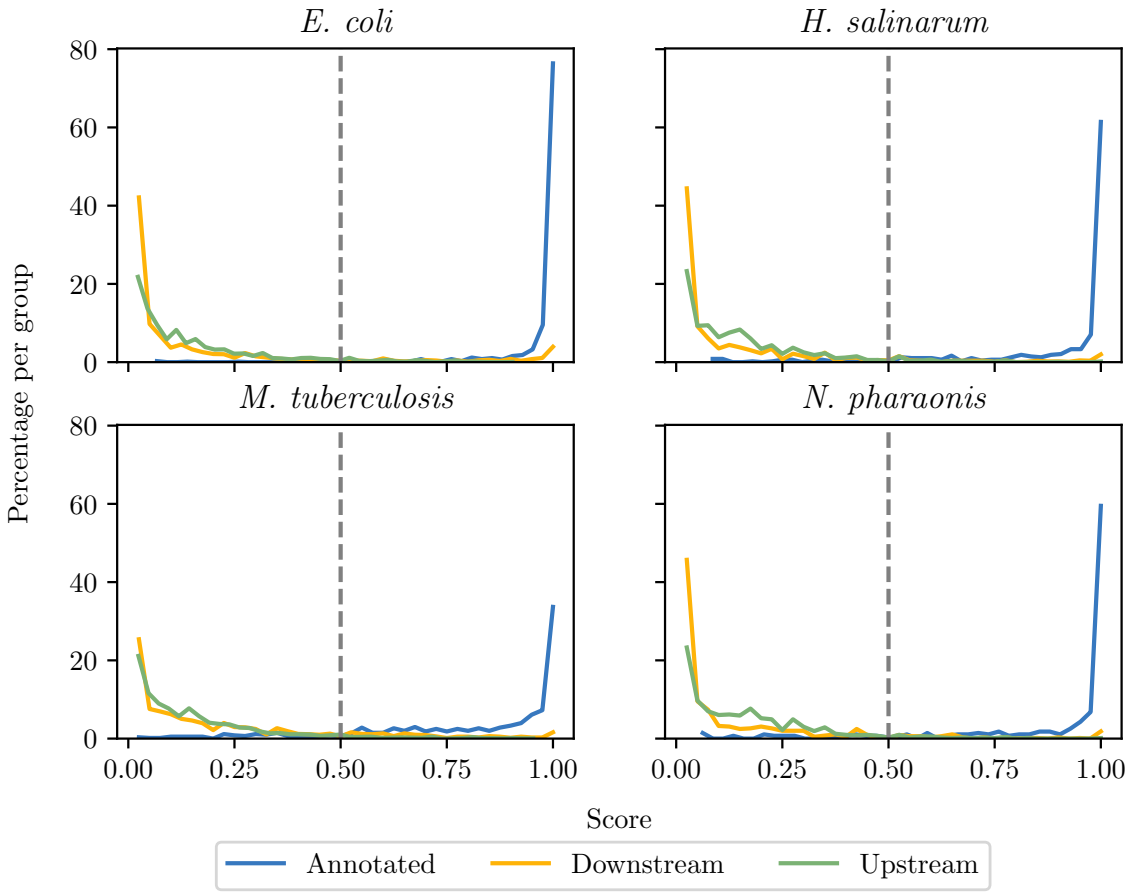


Figure 16: Distribution of 5' identity for verified starts, and upstream and downstream false 5' candidates

```
# setup environment
source config.sh
source install.sh

# download assembly summary files
$bin/download_assembly_summary_py.sh --database refseq --pf-output
    $metadata/refseq_assembly_summary.txt

# construct taxonomy tree
pd_work=$tmp/tree
mkdir -p $pd_work

pf_tree=$pd_work/tree.pkl

$bin/download_taxonomy_dump_py.sh --pd-output $metadata/taxdump
$bin/build_taxonomy_tree_py.sh --pf-nodes-dmp $metadata/taxdump/nodes.dmp --pf-names-dmp
    $metadata/taxdump/names.dmp --pf-tree $pf_tree

# download genomes for clade
clade=Halobacterium
dn_cl=halobacterium
pf_assembly_summary=$metadata/refseq_assembly_summary.txt
pf_t_list=$lists/refseq_${dn_cl}.list

$bin/download_genomes_for_clade_py.sh --pf-tree $pf_tree --pf-assembly-summary
    $pf_assembly_summary --clade-id $clade --clade-id-type "name_txt"
    --favor-assembly-level-order --genomes-per-taxid 1 --pf-output-list $pf_t_list

# extract labeled sequences for targets
pd_work=$tmp/build_database/${dn_cl}

pf_faa=$pd_work/refseq_${dn_cl}.faa
pf_db=$db/refseq_${dn_cl}.dmnd
pf_db_index=$db/index.csv      # database locations

$bin/extract_annotated_sequences_py.sh --pf-genome-list $pf_t_list --pf-output $pf_faa

# build database
$bin/build_blast_db_py.sh --pf-sequences $pf_faa -pf-db $pf_db

# add path to database into database index file
echo -e "Clade,pf-db\n${clade},${pf_db}" > $pf_db_index

[[ -f $pf_faa ]] & rm $pf_faa

cd $base

# run sbsp
pf_sbsp_options=$config/sbsp_0.list
$bin/sbsp_on_genome_list_py.sh --pf-q-list $lists/verified_hsalinarum.list --pd-work
    $runs --pf-sbsp-options $pf_sbsp_options --pf-db-index $pf_db_index
```

```
# the prediction file 'sbsp.gff' will be located in the directory
    $runs/GENOME/sbsp/sbsp.gff
# in this case GENOME is the id for H. salinarum, found in $lists/verified_hsalinarum.list
```

11 Using existing databases

As previously mentioned, all databases must be constructed using the `$bin/extract_annotated_sequences_py.sh` and `$bin/build_blast_db_py.sh` scripts. This is because, for now, the format of the database is strict. This will be relaxed in a later release.

We have provided some example databases. One containing all RefSeq annotated Archaea genomes, and another for RefSeq annotated Actinobacteria genomes. The below shows how to use them for SBSP prediction (example is shown for H. salinarum and Archaea):

```
# CHANGE THIS TO POINT TO DATABASE FILE. USE ABSOLUTE PATHS ONLY
clade=Archaea
pf_db=PATH-TO-DB.dmnd
pf_db_index=$db/index.csv      # database locations

# This file contains the list of query genomes. See existing examples in $lists for
    samples.
# IMPORTANT: the genome and RefSeq labels must already be download and placed in
# $data/GENOME_NAME/sequence.fasta and $data/GENOME_NAME/ncbi.gff
pf_q_list=$lists/verified_hsalinarum.list

# Path of database should be in index.
echo -e "Clade,pf-db\n${clade},${pf_db}" > $pf_db_index

# run sbsp (in this example, we're only running on verified genes (from verified.gff)).
# If --fn-q-labels is not provided, the program takes initial labels from ncbi.gff
pf_sbsp_options=$config/sbsp_defaults.list
$bin/sbsp_on_genome_list_py.sh --pf-q-list $lists/verified_hsalinarum.list --pd-work
    $runs --pf-sbsp-options $pf_sbsp_options --pf-db-index $pf_db_index --fn-q-labels
    verified.gff

# the prediction file 'sbsp.gff' will be located in the directory
    $runs/GENOME/sbsp/sbsp.gff
# in this case GENOME is the id for GENOME_NAME, found in $pf_q_list
```