DaisySuite Documentation

Release 1

Enrico Seiler, Kathrin Trappe, Tobias Marschall, Bernhard Y. Rena

May 10, 2019

Contents

DaisySuite is a mapping-based workflow for analyzing horizontal gene transfer (HGT) events in bacterial data. The Next Generation Sequencing (NGS) input is processed in two major steps. First, DaisyGPS identifies possible acceptor and donor candidates. Second, Daisy determines exact HGT-regions for the acceptor/donor pairs.

CHAPTER 1

Getting started

You can either install DaisySuite with *Conda* or *git*

CHAPTER 2

Citation

- Trappe, K., Marschall, T., Renard, B.Y. “Detecting horizontal gene transfer by mapping sequencing reads across species boundaries”. Bioinformatics. 2016
- Seiler, E., Trappe, K., Renard, B.Y., Marschall, T. “Where did you come from, where did you go: Enhancing Metagenomic Approaches for Pathogen Identification”. biorxiv. 2018

CHAPTER 3

Tools used

- Bioconda: Bioconda website
- Biopython: Cock, P.A., Antao, T., Chang, J.T., Chapman, B.A., Cox, C.J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B. and de Hoon, M.J.L. (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics
- bwa: Li, H. and Durbin, R. (2009) Fast and accurate short read alignment with Burrows-Wheeler Transform. Bioinformatics.
- Conda: Conda website
- Gustaf: Trappe, K., Emde, A., Ehrlich, H., Reinert, K. (2014) Gustaf: Detecting and correctly classifying SVs in the NGS twilight zone. Bioinformatics.
- Laser: Marschall, T. and Schönhuth., A. (2013) LASER: Sensitive Long-Indel-Aware Alignment of Sequencing Reads. arXiv..
- Mason2: Holtgrewe, M. (2010) Mason – A Read Simulator for Second Generation Sequencing Data. Technical Report FU Berlin.
- MicrobeGPS: Lindner, M.S., Renard, B.Y. (2015) Metagenomic Profiling of Known and Unknown Microbes with MicrobeGPS. PLoS ONE
- NumPy: van der Walt, S., Colber, S. and Varoquaux, G. (2011) The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering.
- Pandas: McKinney, W. (2010) Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference.
- pysam: pysam website
- sak: Github
- samtools: Li H., Handsaker B., Wysoker A., Fennell T., Ruan J., Homer N., Marth G., Abecasis G., Durbin R. and 1000 Genome Project Data Processing Subgroup (2009) The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics.
- SciPy: Jones, E., Peterson, P., et al. (2001) SciPy: Open Source Scientific Tools for Python

- Snakemake: Köster, J. and Rahmann, S. (2012) Snakemake - scalable bioinformatics workflow engine. Bioinformatics.
- Stellar: Kehr, B., Weese, D., Reiner, K. (2011) STELLAR: fast and exact local alignments. BMC Bioinformatics.
- Yara: Siragusa, E. (2015). Approximate string matching for high-throughput sequencing. PhD Dissertation, Free University of Berlin.

CHAPTER 4

Contact

Issue Tracker Enrico Seiler Kathrin Trappe

4.1 Install with Conda

4.1.1 Installing DaisySuite

We recommend the usage of Conda to install DaisySuite. If you prefer to not use Conda, you can find instructions in the *Install with git* chapter.

To install Conda under Linux 64bit, run

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh
```

and follow the instructions. Please visit the Miniconda homepage for further information.

Next, add the bioconda channel to your conda installation.

```
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
```

Once Conda is available, DaisySuite can be easily installed via

```
conda install daisysuite
```

Note that older or newer versions of already on your system existing tools, e.g. bwa, will be installed according to the specified requirements for DaisySuite in the Conda package. To install DaisySuite in a new environment and thereby not altering any of your existing installations, use

```
conda create -n daisysuite_env daisysuite
```

where `-n daisysuite_env` specifies the environment name and can be chosen freely. Use `source activate daisysuite_env` to activate your new environment and `source deactivate` to exit the environment.

4.1.2 Additional dependency for Laser

Laser uses bwa for structural variation analysis and requires the additional bwa perl script `xa2multi.pl` that is usually not installed with bwa, but is available in the bwa github. `xa2multi.pl` needs to be present in the `$PATH` variable in order to use Laser, for example:

```
mkdir ~/bin
cd ~/bin
wget https://raw.githubusercontent.com/lh3/bwa/master/xa2multi.pl
export PATH=~/bin:$PATH
```

4.1.3 Setting DaisySuite up

You can automatically download and create all required data by running `DaisySuite_setup <dir>`. This will put the NCBI database and corresponding indices into the directory `<dir>`. Alternatively, the requirements are explained in the *Database requirements section*. Yara needs up to 1 TB of temporary disk space to create its index. Currently, the setup script will fail if one of the steps fails. In this case, consider running the individual steps one by one (see the Database creation section).

4.2 Install with git

4.2.1 Installing DaisySuite

If you prefer to use Conda, you can find instructions in the *Install with Conda* chapter.

To use DaisySuite, the following dependencies need to be satisfied and globally available:

- bedtools
- biopython
- bwa
- clever-toolkit (Laser)
- gustaf
- mason2
- pandas
- pysam
- sak
- samtools
- scipy
- snakemake
- stellar
- yara

To install DaisySuite, run

```
git clone https://gitlab.com/eseiler/DaisySuite.git
cd DaisySuite
chmod +x DaisySuite*
```

For easy access, you might want to add the DaisySuite directory to your PATH variable, e.g.

```
export PATH=~/DaisySuite/:$PATH
```

4.2.2 Additional dependency for Laser

Laser uses bwa for structural variation analysis and requires the additional bwa perl script `xa2multi.pl` that is usually not installed with bwa, but is available in the bwa github. `xa2multi.pl` needs to be present in the `$PATH` variable in order to use Laser, for example:

```
mkdir ~/bin
cd ~/bin
wget https://raw.githubusercontent.com/lh3/bwa/master/xa2multi.pl
export PATH=~/bin:$PATH
```

4.2.3 Setting DaisySuite up

You can automatically download and create all required data by running `DaisySuite_setup <dir>`. This will put the NCBI database and corresponding indices into the directory `<dir>`. Alternatively, the requirements are explained in the *Database requirements section*.

4.3 Database requirements

You can automatically download and create all required data by running `DaisySuite_setup <dir>`. This will put the NCBI database and corresponding indices into the directory `<dir>`. Alternatively, the individual requirements are explained in this section.

DaisySuite requires following data:

- NCBI RefSeq
- bwa index of the NCBI RefSeq
- yara index of the NCBI RefSeq
- MicrobeGPS taxonomy files

We will provide example code snippets to solving each one of the requirements.

4.3.1 NCBI RefSeq

First, download all NCBI entries that are tagged as "complete genomes" into a directory of your choice, e.g.

```
### Download NCBI database
mkdir ncbi
wget ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq/bacteria/assembly_summary.txt
wget --directory-prefix ncbi -q ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/367/
↪745/GCF_000367745.1_ASM36774v1/GCF_000367745.1_ASM36774v1_genomic.fna.gz
```

```
cat assembly_summary.txt | \
awk '{FS="\t"} !/^#/ $12 ~ "Complete Genome" {print $20}' | \
sed -r 's|(ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/[0-9]*/[0-9]*/[0-9]*/)(GCF_.
↪+)|\1\2/\2_genomic.fna.gz|' | \
xargs -n 1 -P 12 wget --directory-prefix ncbi -q
rm assembly_summary.txt
```

Note that we download an additionial file in line 4. It contains the acceptor reference of one of our datasets and is wrongly labeled as not complete, hence the manual download.

4.3.2 Preprocessing NCBI

Next, we want to

- extract the downloaded references
- split references containing multiple sequences
- remove plasmid references
- rename the fasta files to <accession.version>.fasta
- get a list of all accessions for future use
- merge the fasta files (while keeping the originals)
- gzip all fasta files

First, we need to extract the references.

```
### Extract NCBI database
gzip -d /ncbi/*_genomic.fna.gz
```

The pipeline requires the fasta files to only contain a single reference. This can be done, for example, with the python script `split_fasta.py` in the `data/scripts/src` directory of your DaisySuite installation.

```
### Split fasta files
for f in ncbi/*.fna;
do
    src/split_fasta.py $f
done
```

```
#!/usr/bin/python3
import os

def split_multifasta(fname):
    count = 0
    g = None
    with open(fname, 'r') as f:
        comp = fname.split('.')
        for line in f:
            if '>' in line:
                if g is not None:
                    g.close()
                g = open('{}_split{}.{}'.format('.'.join(comp[:-1]), count, comp[-1]),
↪ 'w')
                count += 1
                g.write(line)
            else:
```

```
                g.write(line)
    os.remove(fname)

if __name__ == '__main__':
    import sys
    import os
    split_multifasta(os.path.realpath(sys.argv[1]))
```

The DaisyGPS part of DaisySuite currently can not detect plasmids as potential HGT donors. The plasmids need to be excluded from the database and the remaining references should be named <accession.version>.fasta

```
### Rename fasta files and exclude plasmids
for f in ../ncbi/*.fna;
do
    if [[ $(head -n 1 $f) != *"Plasmid"* ]] && [[ $(head -n 1 $f) != *"plasmid"* ]]
    then
        accession=$(sed 's/>\([A-Za-z0-9\._]*\) .*/\1/' <(head -n 1 $f))
        new=$(dirname $f)/$accession.fasta
            mv $f $new
    else
        rm -f $f
    fi
done
```

For the creation of a taxonomic file for MicrobeGPS we need a list of all references included in our database:

```
### Get accession list
for i in ../ncbi/*.fasta*;
do
    n=$(sed 's/\(.*\)\.fasta.*/\1/' <(basename $i))
    cat <(echo $n) >> acc.txt
done
```

Most mapper indexer accept only one fasta file containing all references, so we merge our sequences. We also compress the resulting fasta to save space.

```
### Merge fasta files
cat ncbi/*.fasta > ncbi/reference.fasta

### Gzip fasta files
gzip ncbi/*.fasta
```

4.3.3 Taxonomy

MicrobeGPS is used in DaisyGPS and leverages taxonomic information for the profiling of the sample. Therefore, several taxonomic files are needed:

- NCBI's nodes.dmp
- NCBI's names.dmp
- a catalog (bact.catalog) containing accession.taxid, taxid and name of each reference present in the NCBI database

The `nodes.dmp`, `names.dmp` and `bact.catalog` must be saved in the `data/microbeGPS/data` folder of your DaisySuite installation.

```
### Get names.dmp and nodes.dmp
wget ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz
tar xfz taxdump.tar.gz names.dmp nodes.dmp
mv names.dmp data/microbeGPS/data/names.dmp
mv nodes.dmp data/microbeGPS/data/nodes.dmp
rm taxdump.tar.gz
```

To create the catalog file, you can use the `createMinimalMGPSCatalog.py` in the `data/scripts/src` directory of your DaisySuite installation. It requires the NCBI nucl_gb.accession2taxid file, names.dmp and the list of all accessions in the NCBI database generated in the *preprocessing step*.

```
### Get catalog
wget -T 1800 -qO- ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/accession2taxid/nucl_gb.
↪accession2taxid.gz | gzip -dc > nucl_gb.accession2taxid
data/src/createMinimalMGPSCatalog.py nucl_gb.accession2taxid data/microbeGPS/data/
↪names.dmp data/microbeGPS/data/bact.catalog -ref acc.txt
rm nucl_gb.accession2taxid
```

The accession list is now no longer needed and can be deleted.

```
### Remove accession list
rm acc.txt
```

4.3.4 Indices

In a last step, we need to generate the yara or bwa index (one is sufficient to run DaisySuite). During development we found Yara to be more sensitive regarding the mapping, but it needs up to 1 TB of temporary disk space to build its index.

```
### Create bwa index
bwa index ncbi/reference.fasta.gz -p bwa/bwa_index

### Create yara index
yara_indexer ncbi/reference.fasta.gz -o yara/yara_index
```

After this, we can delete the merged reference file.

```
### Remove merged fasta file
rm ../ncbi/reference.fasta.gz
```

4.4 DaisySuite configuration

The configuration of DaisySuite is done by a yaml file. You can generate a template containing all options with explanation by using

```
DaisySuite_template .
```

to create a template in the current working directory. Additionally, the options are explained on this page.

4.4.1 General configuration

There are three parameters that determine which steps of the pipeline are run (read simulation, DaisyGPS, Daisy). Possible values are `true` and `false`. To run, for example, DaisyGPS and Daisy on an existing read dataset, use:

```
sim: false
daisygps: true
daisy: true
```

Also, an output directory must be defined via the `outputdir` variable:

```
outputdir: 'path/to/output/directory'
```

If `sim` is set to `true` this directory can be empty, otherwise the reads are expected to be in the `read` directory of the output directory. The reads must be named `<sample>.1.fq` and `<sample>.2.fq`, see also Using DaisySuite.

4.4.2 Simulation configuration

The simulation needs the NCBI folder created by the setup (see Using DaisySuite), e.g.:

```
ncbidir: 'path/to/DaisySuite/data/ncbi'
```

You can futher specify how many simulations are to be run, the coverage and the read length:

```
simulates: 1
coverage: 100
readlength: 150
```

This will create one simulated read data set where the reference is covered about 100 times by reads with an average length of 150.

You can also choose to not incorporate any HGT event, e.g. to test for specificity:

```
negative: true
```

Furthermore, there are various parameters to configure the read simulation done by mason, e.g. SNP rate:

```
# Acceptor: Mason variator SNP rate
a_snp: 0.01
# Acceptor: Mason variator small indel rate
a_sir: 0.001
# Acceptor: Mason variator sv indel rate
a_svir: 0.00001
# Donor: Mason variator SNP rate
d_snp: 0.001
# Donor: Mason variator small indel rate
d_sir: 0.001
# Donor: Mason variator max small indel size
d_msis: 4
```

4.4.3 DaisyGPS configuration

You can select a mapper (bwa or yara) and the number of threads used by the mapper:

```
bwa: false
threads: 20
```

Those settings are shared with Daisy.

According to the choice of your mapper, you need to specify the location of the index. The indices are generated during the setup (see Using DaisySuite):

```
yara_index: 'path/to/yara_index'
bwa_index: 'path/to/bwa_index'
```

Note that you can specify both indices, the pipeline will pick the index according to your chosen mapper.

You also need to specify the NCBI directory as described in the *Simulation section*.

You can select taxons per sample that should not be included in the candidate selection. You can either blacklist single taxons, a whole species or all children of a taxon:

```
taxon_blacklist:
  - sample1:
    - 672612
  - sample2:
    - 726312

species_blacklist:
  - sample2:
    - 1270

parent_blacklist:
  - sample2:
    - 3173
```

You can also opt to only report candidates that are classified more specific than species level, i.e. if a taxon has the same taxonomic ID as the corresponding species taxonomic ID, the taxon is not reported. If the resulting candidates list would be empty with this filter activated, the filter is ignored.

```
filter_species: true
```

Furthermore, you can set how many acceptors, donors and acceptor-like donors are reported:

```
number_acceptors: 2
number_donors: 3
number_accdons: 2
```

4.4.4 Daisy configuration

The mapper and threads choices are shared with *DaisyGPS*.

You can either use a sensitive mode (using Stellar and Gustaf) or less sensitive mode (using laser).

```
sensitivemode: true
```

In case you are using the sensitive mode, the reads are mapped and filtered for not properly aligned pairs. There is an option to only filter for reads where at least one mate is unmapped, if the filtering for not properly aligned pairs should yield more than 750000 entires. This will decrease the runtime at the cost of decreased sensitivity.

```
samflag_filter: true
```

Another option for the sensitive mode is to define the needed number of reads that support a breakpoint for Gustaf. The default is 2 and increasing it will lead to a more strict search for breakpoints. This option is set per sample.

```
gustaf_st:
  - sample1:
    - 4
```

Daisy can also search against a phage database. An empty entry will result in no search against the phage DB. If you wish to search against a database, inlclude the full path to the fasta file.

```
phage: ''
```

Furthermore, you can set parameters regarding the HGT detection, i.e. the minimum/maximum HGT size and the needed support by sampling.

```
hgt_min: 100
hgt_max: 55000
hgt_sens: 90
```

4.4.5 Command line help

`DaisySuite -h` provides a short command overview

```
DaisySuite Pipeline (powered by Snakemake)

Usage: DaisySuite --configfile FILE [Snakemake options]

 Useful Snakemake parameters:
   -j, --cores            number of cores
   -k, --keep-going       go on with independent jobs if a job fails
   -n, --dryrun           do not execute anything
   -p, --printshellcmds   print out the shell commands that will be executed

 Full list of parameters:
   --help                 show Snakemake help (or snakemake -h)
```

4.5 Using DaisySuite

4.5.1 Preparation

If you want to simulate reads and then analyze them, you only need to specify the outputdir and your database directories in the config. If you already have (gzipped) fastq reads, these files need to be present within the `` `reads` `` folder in the outputdir. Furthermore, the reads need to follow a specific naming scheme: `{sample}.1.fq` for single end reads `{sample}.1.fq` and `{sample}.2.fq` for paired end reads {sample} is an arbitrary name for the read data set The reads can be either in plain fastq format or gzipped (fq.gz). Please note that Laser cannot be used with single end reads.

The read names (the read names within the fastq file) can not contain a slash ('/') symbol. For example, simulated.1/1 will not work. (remove, e.g. by using `` `sed -i 's/\(@simulated\.[0-9]*\)\/[0-9]*/\1/' <fq-file>` ``)

4.5.2 Running with known acceptors and donors

If you have a read dataset and know possible acceptor and donor organisms, you can skip the DaisyGPS and directly use Daisy to detect HGT events. To do this, you need to set `sim` and `daisygps` in the config to `false` and `daisy` to `true`. Now create the following directory structure: . – outputdir | – reads | – {sample}.1.fq.gz *read file 1* | – {sample}.2.fq.gz *read file 2* | – candidates | – {sample}_acceptors.fa *Sequences for all acceptor candidates* | – {sample}_acceptors.txt *Accession.Version for all acceptor candidates* | – {sample}_donors.fa *Sequences for all donor candidates* | – {sample}_donors.txt *Accession.Version for all donor candidates*

The `reads` folder contains your read dataset (paired or single end) and is prefixed with `sample`. `candidates/{sample}_acceptors.fa` is the concatenation of all the sequences of the acceptor candidates (a (multi-)FASTA file). `candidates/{sample}_acceptors.txt` lists all the accesions that appear in `candidates/{sample}_acceptors.fa` (one accession per line). Analogously, you have to provide `{sample}_donors.fa` and `{sample}_donors.txt`.

4.5.3 Example directory

. – outputdir | – reads | – {sample1}.1.fq.gz | – {sample1}.2.fq.gz | – {sample2}.1.fq.gz | – {sample2}.2.fq.gz | – {sample3}.1.fq.gz

All samples inside the outputdir will be run.

4.5.4 Config file

All necessary parameters are set in a configfile. A template file can be generated by running `DaisySuite_template` . For further information, visit the *Configuration section*.

4.5.5 Running a job

This pipeline uses Snakemake and therefore supports all parameters implemented by Snakemake.

```
DaisySuite --configfile config/example.yaml
```

See `DaisySuite -h` or the *Command line help section* for additional options.

4.5.6 Output

The results of DaisyGPS are in the `candidates` directory. The results of Daisy are in the `hgt_eval` directory.

. – outputdir | – log-{timestamp}.txt *logfile containing information about the run* | – reads | – {sample1}.1.fq.gz *read file 1* | – {sample1}.2.fq.gz *read file 2* | – candidates | – {sample1}_acceptors.fa *Sequences for all acceptor candidates* | – {sample1}_acceptors.txt *Accession.Version for all acceptor candidates* | – {sample1}_candidates.tsv *All reported candidates including several metrics* | – {sample1}_complete.tsv *All mapped references including several metrics* | – {sample1}_donors.fa *Sequences for all donor candidates* | – {sample1}_donors.txt *Accession.Version for all donor candidates* | – hgt_eval | – {sample1}.fasta *Sequences of filtered horizontally transferred regions* | – {sample1}.tsv *All hgt regions* | – {sample1}.vcf *All filtered hgt regions*

Additonally, there are many intermediate directories, which can be kept with `--nt`.

Directory	Description
bwa/yara	[Daisy] contains index for acceptor/donor candidates
candidate_mapping	[Daisy] contains mapping of candidates against ncbi
fasta_npa	[Daisy] contains not properly aligned reads in fasta format
fastq_npa	[Daisy] contains not properly aligned reads in fastq format
gustaf	[Daisy] contains results of gustaf
hgt_candidates	[Daisy] contains hgt candidates generated in the hgt evaluation step
hgt_eval_pair	[Daisy] single result files, merged files are found in hgt_eval
joined	[Daisy] contains joined sequences of acceptor and donor candidates
mapping	[DaisyGPS] contains mapping of reads against ncbi
mgps	[DaisyGPS] contains results of MicrobeGPS
npa_joined	[Daisy] contains merged fasta_npa files
process	[DaisyGPS] contains intermediate results of candidate detection
sort_mapping	[Daisy] contains sorted alignment file from candidate_mapping
sort_npa	[Daisy] contains sorted fasta file from npa_joined
stellar	[Daisy] contains results of stellar

4.5.7 Explanation

The objective of the DaisyGPS workflow is to identify possible acceptor (organism that receives genetic information) and donor (organism that the information is transferred from) candidates given reads of a potential HGT organism. The HGT organism's genome consists mainly of the acceptor genome. When the reads of the HGT organism are mapped against the acceptor reference, most reads should map properly. Therefore a high and continuous mapping coverage pattern of the acceptor genome can be expected.

In contrast to that, only a small part of the donor genome is present within the HGT organism's genome, hence only a small fraction of the reads should map against the donor reference and then only within a zoned part (i.e. the part that has been transferred). This results in a discontinuous mapping coverage pattern where only a small part of the reference shows a high mapping coverage.

Given only the reads of the HGT organism, the acceptor and donor candidate identification problem is similar to what is done in metagenomic profiling. We hence use the metagenomic profiling tool MicrobeGPS to gather a profile of our given HGT organism and mapping coverage metrics. MicrobeGPS fits our requirements as it uses metrics that can be adapted to represent acceptor and donor attributes. The candidates are ranked by this score and a list with putative acceptor and donor candidates is generated. These acceptor and donor candidates can then be further analysed with tools like Daisy.

MicrobeGPS

MicrobeGPS is a metagenomic profiling tool. Given a set of references and sequencing reads from a sample it uses references to describe the potential (new) organisms in the sample. That is, it uses available references to model the composition of the organisms present in the sample instead of directly assigning reference organisms to characterize the sample. To do this, MicrobeGPS developed metrics to compare the genomes in the sample with the reference genomes. In doing so, MicrobeGPS computes a genome coverage and a metric that corresponds to whether a mapping pattern is continuous or not. Therefore, it is possible to use MicrobeGPS to identify not only acceptor but also the donor by leveraging those metrics. We generated the alignment file required for MicrobeGPS by mapping the HGT organism's reads against the NCBI RefSeq using bwa and yara.

To to describe the similarity between an organism found in the sample and the corresponding reference genome based on read evidence, MicrobeGPS facilitates a genomic distance measurement, namely the Genome Dataset Validity (GDV, validity for short). It describes the fraction of the reference genome for which there is read evidence. A taxonomic position then can be estimated by using the GDV as a measurement for the distance between an organism found in the sample and the closest reference genome. To account for possible biases arising from the use of short NGS reads, the read mapping is modelled as a mixture distribution:

$f(x|\overline{\alpha}, \lambda) = \alpha_1 \cdot z(x) + \alpha_2 \cdot \ P(x|\lambda) + \alpha_3 \cdot T_\alpha(x)$

where $z(x)$ (zero distribution) and $P(x)$ (Poisson distribution) form a zero-inflated Poisson distribution that allows to account for zero coverage regions. The tail distribution accounts for differences between reads and reference genome. α_1, α_2 and α_3 are the mixture coefficients of the different distributions, respectively. Since $z(x)$ represents the zero coverage regions, the validity can be easily calculated by $1 - \alpha_1$

Secondly, another key point of interest is to determine how evenly the reads are distributed. In MicrobeGPS, a Kolmogorov-Smirnov (KS)-Test is used to compare the actual read distribution with an theoretical read distribution that is generated by placing the reads uniformly across the genome, i.e. after having seen $x\%$ of the genome also $x\%$ of the reads should have been seen. The heterogeneity (note: called homogeneity in the original work) is then the KS-Statistic:

$D_n = \sup_x \mid F_n(x) - F(x) \mid$

D_n is the KS statistic, $F_n(x)$ is the empirical distribution function (what we expect), $F(x)$ the given cumulative distribution function (what we have)

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} I_{[-\infty,x]}(X_i) I_{[-\infty,x]}(X_i) = \begin{cases} 1 & X_i < x00 \\ else \end{cases}$$

Validity and Heterogeneity describe how much of the reference is covered and how evenly the reads are distributed, respectively. Therefore, they can be used to classify acceptor and donor candidates for an HGT event.

Acceptor and donor classification

For an acceptor, the mapping coverage is relatively high and homogeneous. Hence, we expect a high validity score and a low heterogeneity score. In contrast to that, we expect a low validity and high heterogeneity for donors. These assumptions can be used to define a property score that reflects how much the attributes of a given organism matches the properties of an acceptor or donor:

$property = validity - heterogeneity$ with $property \in (-1, 1]$

Therefore, the value for a completely covered acceptor with uniform read distribution would approach $+1$. Likewise, the value for a donor that is only covered in a small region would approach -1. We tested weighted scores that include additional information such as the number of unique reads or *k-mer* counts (taken, e.g., directly from Kraken), but they did not improve the classification of suitable candidates. For both mentioned criteria, a high count did not correlate with the candidate being an acceptor or donor (data not shown). There is, however, a high evidence by sheer read numbers for acceptors:

$propertyscore = \#mapped\_reads/\#total\_reads * property$

Where $\#mapped\_reads/\#total\_reads$ is the fraction of all mapped reads that mapped to the specific acceptor candidate.

For the donor, however, the size of the transferred region is not known in advance, hence, we do not expect a specific read number evidence and therefore omit the weighting, which yields $property$ as metric. We select the acceptors with the highest property scores (approximating 1) and the donors with the lowest property (approximating -1) as candidates.

In case acceptor and donor are very similar, the donor might not express the attributes we are looking for. In particular, the donor might have a significant read number evidence arising from acceptor reads also mapping to the donor.

As a result, the property score will likely indicate an acceptor instead of a donor (approximating $+0$), while still being relatively low compared to the property of other acceptors.

To account for this, we also classify candidates with a $propertyscore > 0$ as possible acceptor-like donors. The entries with the lowest property among them are selected as acceptor-like donors.

For more information and publications, see Citations

4.6 DaisySuite example

To run DaisySuite on an example dataset, first copy the example into a directory of your choice by running

```
DaisySuite_example .
```

to copy the folder `example` into the current working directory.

Next, you need to edit the following parameters in the example/example.yaml:

- outputdir (full path to example/output/)
- ncbidir
- bwa (in case you are using bwa)
- yara_index or bwa_index

Finally, you can run DaisySuite:

```
DaisySuite --configfile example/example.yaml
```

You can also use multiple threads by adding `--cores <thread_number>`, e.g. `--cores 10`, to the command.

4.6.1 DaisyGPS results

You will find the acceptor *Escherichia coli str. K-12 substr. DH10B* [NC_010473.1] and the donor *Helicobacter pylori* [NZ_AP014710.1] in the `example/output/candidates/sim1HP_candidates.tsv` file.

Table 4.1: DaisyGPS Results

Type	Name	Accession.Version	TaxID	Parent TaxID	Species TaxID	Abundance	Num. Reads	Unique Reads	Coverage	Validity	Homogeneity	Mapping Error	Property Score	Property
Acceptor	Escherichia coli str. K-12 substr. DH10B	NC_0[illegible]	[illegible]	[illegible]	562	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Acceptor	Escherichia coli K-12	NZ_C[illegible]	[illegible]	562	562	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Donor	[Haemophilus] ducreyi	NZ_C[illegible]	[illegible]	[illegible]	730	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Donor	Salmonella enterica subsp. enterica serovar Anatum str. USDA-ARS-USMARC-1676	NZ_C[illegible]	[illegible]	[illegible]	28901	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Donor	Klebsiella oxytoca KONIH1	NZ_C[illegible]	[illegible]	[illegible]	571	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Donor	Helicobacter pylori	NZ_A[illegible]	[illegible]	[illegible]	210	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Acceptor-like Donor	Escherichia coli	NZ_C[illegible]	[illegible]	[illegible]	562	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

4.6.2 Daisy results

Furthermore, you will find the base pair positions of the transfer in `example/output/hgt_eval/sim1HP.vcf`. Bases 1322000 to 1350000 of the donor have been inserted at base 1120262 of the acceptor. This is indicated by two breakpoints in the vcf, one representing the beginning of the insert (acceptor 1120261, donor 1322000) and one representing the end of the insert (acceptor 1120263, donor 1350000).

The `example/output/hgt_eval/sim1HP.tsv` also provides a more intuitive representation of putative transferred regions, but please note that those candidates have not been filtered by the sampling values.

Listing 4.1: Daisy TSV header

```
#AN: Acceptor name
#DN: Donor name
#AS: Acceptor start position
#AE: Acceptor end position
#DS: Donor start position
#DE: Donor end position
#MC: Mean coverage in region
#Split: Total number split-reads per region (including duplicates!)
#PS-S: Pairs spanning HGT boundaries
#PS-W: Pairs within HGT boundaries
```

```
#Phage: PS-S and PS-W reads mapping to phage database
#BS:MC/PS-S/PS-W: Percent of bootstrapped random regions with MC/PS-S/PS-W smaller
↪than candidate
```

Table 4.2: Daisy Results TSV

AN	DN	AS	AE	MC	BS:MC	DS	DE	MC	Split	PS-S	PS-W	Phage	BS:MC	BS:PS-S	BS:PS-W
NZ_CP010445.1	NZ_AP014710.1	1880235	1880237	4.00	7	1322001	1350009	94.62	152	182	8712	0.0000	100	100	100
NZ_CP010445.1	NZ_CP015434.1	3904873	3904886	40.54	3	1149281	1269573	0.41	871	156	884	0.0000	100	100	100
NZ_CP010445.1	NZ_CP015434.1	3904873	3916007	97.63	20	1256261	1269573	29.68	1571	108	258	0.0000	100	100	100
NZ_CP010445.1	NZ_CP015434.1	3904885	3916007	97.69	18	1149271	1256261	8.06	253	43	279	0.0000	100	99	100
NC_010473.1	NZ_AP014710.1	1120261	1120263	3.00	3	1322001	1350009	94.62	154	182	8712	0.0000	100	100	100

Listing 4.2: Daisy VCF header

```
##fileformat=VCFv4.2
##source=DAISY
##INFO=<ID=EVENT,Number=1,Type=String,Description="Event identifier for breakends.">
##contig=<ID=NC_010473.1>
##contig=<ID=NZ_CP010445.1>
##contig=<ID=NZ_CP015434.1>
##contig=<ID=NZ_CP014620.1>
##contig=<ID=NZ_CP008788.1>
##contig=<ID=NZ_AP014710.1>
##contig=<ID=NZ_CP016182.1>
```

Table 4.3: Daisy Results VCF

CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT
NZ_CP010445.1	1880235	BND_1_1	A	A[NZ_AP014710.1:1322001[	152	PASS	SVTYPE=BND;EVENT=HGT1	.	1
NZ_CP010445.1	1880237	BND_1_2	G	]NZ_AP014710.1:1350009]G	182	PASS	SVTYPE=BND;EVENT=HGT1	.	1
NZ_CP010445.1	3904873	BND_1_1	T	T[NZ_CP015434.1:1149281[	871	PASS	SVTYPE=BND;EVENT=HGT1	.	1
NZ_CP010445.1	3904886	BND_1_2	C	]NZ_CP015434.1:1269573]A	156	PASS	SVTYPE=BND;EVENT=HGT1	.	1
NC_010473.1	1120261	BND_1_1	A	A[NZ_AP014710.1:1322001[	154	PASS	SVTYPE=BND;EVENT=HGT1	.	1
NC_010473.1	1120263	BND_1_2	G	]NZ_AP014710.1:1350009]G	182	PASS	SVTYPE=BND;EVENT=HGT1	.	1