SNAPPy-HIV1-Subtyping

Release 1.0.0

Jul 05, 2019

Contents

SNAPPy's code can be found here

What is SNAPPy

SNAPPy is a Snakemake pipeline for HIV-1 subtyping by phylogenetic pairing. It was build to allow locaal high-throughput HIV-1 subtyping while being resource efficient and scalable. This pipeline was constructed using Snakemake , and it uses MAFFT for multiple sequence alignments, BLAST for similarirty queries, FastTree for phylogenetic inference, and several Biopython modules for data parsing an analysis. For in-depth information on how the tool works please visit the *How it works* section. SNAPPy was design for Linux based operative systems.

Getting Started

- Installation:
  - *Quick install Guide*
  - *Install point-by-point*
- Tutorials:
  - *1: Subtyping with SNAPPy*
  - *2: Alignments with SNAPPy*
  - *3: Result Analysis*
- Commands list:
  - *SNAPPy commands*

Support

- The software is provided "as is", without warranty of any kind, we do our best to keep it updated and bug free
- *FAQ*
- For additional information please contact: INFO (MAIL)

How to Cite

text

CHAPTER 1

Installation

This pipeline was build upon many dependencies and Python code. Therefore there was the need to use a package/environment manager. For this task we highly recommend CONDA . The following installation instructions assume the user wants to install and use CONDA. If you already have any recent conda distribution on your system do not use the *Quick install Guide*, as it will install Miniconda. If you do not which to use CONDA please look at all the dependencies needed to use SNAPPy in the 'environment.yaml' file.

1.1 Quick install Guide

1) Download of clone SNAPPy from here and put it in a desired location
2) Open a terminal and move to the downloaded folder:

```
cd snappy
```

3) Give permission to the snappy installation script to run, and run it:

```
chmod +x snappy_installer_x86_64.sh
./snappy_installer_x86_64.sh
```

This task is time consuming and may take several minutes.

4) **After the SNAPPy's dependencies installation a series of tests will be executed using pytest. Please ensure no errors occur.**

    a) If there was any errors clean the downloaded folder and start from step 1. If the problem persist please try to follow the *Install point-by-point*. If you are still unable to install SNAPPy after this contact us to EMAIL.

5) SNAPPy is now ready to use! When you want to use SNAPPy open a new terminal inside SNAPPy's folder and activate the conda environment by typing:

```
conda activate snappy
```

For a quick start on SNAPPy's usage look at the *Tutorials* section.

1.2 Install point-by-point

1) Open a terminal, download Miniconda3 version 4.6.14 and install it:

```
wget https://repo.continuum.io/miniconda/Miniconda3-4.6.14-Linux-x86_64.sh -O ./
↪miniconda.sh
bash ./miniconda.sh -b -p $HOME/miniconda
```

This task may take several minutes.

2) Download or clone SNAPPy from here and put it in a desired location

3) Move to the downloaded folder:

```
cd snappy
```

4) Activate conda and add it tho the bash paths:

```
source ~/miniconda/etc/profile.d/conda.sh
conda init
```

Note: It is not actually needed to add conda path to the bash path but it makes it easier to use.

5) Create SNAPPy's conda environment:

```
conda-env create -f environment.yaml
```

6) Activate SNAPPy's conda environment:

```
conda activate snappy
```

7) Run the tests to ensure the installation was successful. Please ensure no errors occur:

```
py.test
```

8) SNAPPy is now ready to use! When you want to use SNAPPy open a new terminal inside SNAPPy's folder and activate the conda by typing:

```
conda activate snappy
```

For a quick start on SNAPPy's usage look at the *Tutorials* section.

CHAPTER 2

Tutorials

Please ensure SNAPPy's installation was successful. If you need help regarding SNAPPy's installation please visit the *Installation* section. To use SNAPPy the terminal always needs to be executed from the snappy folder, and the conda environment activated:

```
conda actiavte snappy
```

If the tool is stopping abruptly or behaving unexpectedly ensure it is functional by simply running the tests:

```
py.test
```

To run the tests the conda snappy environment needs to be active. If the tests result in error it is advised to reinstall the tool.

2.1 1: Subtyping with SNAPPy

Disclaimer: This tutorial and Tutorial *2: Alignments with SNAPPy* share the first four steps.

HIV-1 subtyping is the main reason SNAPPY was build. To use SNAPPy the terminal always needs to be executed from the snappy folder, and the conda environment activated:

```
conda actiavte snappy
```

We highly encourage to run the tests before using the pipeline just to make sure the tasks will run as intended and the pipeline was not modified or damaged:

```
py.test
```

1) Look at the content of the snappy folder:

```
ls
```

Expected result:

```
config.yaml  data  environment.yaml  input  LICENSE.txt  README.md  scripts
↪Snakefile  snappy_installer_x86_64.sh  test  test_pipeline.py
```

2) In this tutorial we will focus on the folder 'input' and the file 'config.yaml'. For exploration of the remaining content please look at the *How it works* section. Lets look inside the folder input:

```
ls input
```

Expected result:

```
test_msa.fasta
```

3) As you can see SNAPPy has a file inside the folder 'input'. This is a test file used in this tutorial. For a successful run only one file can be inside the 'input' folder. Therefore, if you want to run an analysis with your data you need to first delete the default file inside the 'input' folder.

4) Now lets look at the 'config.yaml' file:

```
cat config.yaml
```

Expected result:

```
genomic_region:
  'GAG-POL-ENV'
```

5) As you can see the genomic region selected for the SNAPPy's analysis is 'GAG-POL-ENV' which basically consists the HIV-1 regions named GAG, POL, and ENV concatenated (according to HXB2 (K03455) reference genome). To use SNAPPy for subtyping the selected genomic region in the 'config.yaml' file always needs to be 'GAG-POL-ENV'. If for any reason you have changed it (for instance during Tutorial *2: Alignments with SNAPPy*) change it back.

6) In order to subtype using SNAPPy simple type:

```
snakemake subtype_all
```

7) The SNAPPy pipeline will then be executed and at green you should see each task being done. It should terminate with the following line:

```
11 of 11 steps (100%) done
```

8) Now lets see the files and folders created by SNAPPy during the subtyping process:

```
ls
```

9) As you can see the folders 'aligned'; 'blast'; 'trees' and the files 'subtype_results.csv' and 'report_subtype_results.csv' were created. The file 'subtype_results.csv' has the results that we wanted from running this task, the file 'report_subtype_results.csv' has a more extensive description of the tool outputs (more about this in the Tutorial *3: Result Analysis*). The folders 'aligned', 'blast', and 'trees' contain the intermediate files created by SNAPPy during the subtyping process (more about this in the section *How it works*). Now lets look at the file 'subtype_results.csv' (Note: You can use any other text editor or spreadsheet visualization tool.):

```
cat subtype_results.csv
```

10) As you can see in this csv file there is the header of each fasta in the input followed by the result from SNAPPy subtyping. The 'id' numbers refers to the internal identifier used during the pipeline and links to the intermediate files in the 'aligned', 'blast', and 'trees' folders.

11) Lets try to run the exactly same task again:

```
snakemake subtype_all
```

Expected result:

```
Building DAG of jobs...
Nothing to be done.
```

12) Nothing was done because the output that we requested was already built! This is one of the great advantages of using a pipeline software like Snakemake , it goes top down looking for the requested file and the files needed to create it. If it is already there nothing needs to be done.

13) Now lets use the SNAPPy rule that allows us to clean all the outputs from a previous run. Attention!! If you have results that you want to keep change their name or move them to another folder before running the clean-up command:

```
snakemake delete_all_outputs
ls
```

14) We are now back where we started without any output built. Lets run the pipeline but this time lets use more computational resources, namely four cpu threads:

```
snakemake subtype_all --cores 4
```

15) As you probably noticed this time the same process took a lot less time to run, that's because SNAPPy leverages the Snakemake capabilities of parallelizing tasks. This allows SNAPPy to be extremely scalable. For instance if you have accesses to a n core cpu in theory you can use all of them to do subtyping with SNAPPy in one single task.

16) That's it for this tutorial! If you now want to use SNAPPy on your own date don't forget to clean the outputs created during this tutorial and adjust the content of the input folder.

2.2 2: Alignments with SNAPPy

Disclaimer: This tutorial and Tutorial *1: Subtyping with SNAPPy* share the first four steps.

Althought SNAPPy was built for HIV-1 subtyping one of its intermediary tasks is alignment to the reference genome (HXB2 (K03455)). Since SNAPPy is built based on Snakemake we can call intermediary tasks, such as alignment, without running the entire pipeline. Making SNAPPy extremely useful for performing HIV-1 alignments. To use SNAPPy the terminal always needs to be executed from the snappy folder, and the conda environment activated:

```
conda actiavte snappy
```

We highly encourage to run the tests before using the pipeline just to make sure the tasks will run as intended and the pipeline was not modified or damaged:

```
py.test
```

1) Look at the content of the snappy folder:

```
ls
```

Expected result:

```
config.yaml  data  environment.yaml  input  LICENSE.txt  README.md  scripts␣
↪Snakefile  snappy_installer_x86_64.sh  test  test_pipeline.py
```

2) In this tutorial we will focus on the folder 'input' and the file 'config.yaml'. For exploration of the remaining content please look at the *How it works* section. Look inside the folder input:

```
ls input
```

Expected result:

```
test_msa.fasta
```

3) As you can see SNAPPy has a file inside the folder 'input'. This is a test file used in this tutorial. For a successful run only one file can be inside the input folder. Therefore, if you want to run an analysis with your data you need to first delete the default file inside the input folder.

4) Now lets look at the 'config.yaml' file:

```
cat config.yaml
```

Expected result:

```
genomic_region:
  'GAG-POL-ENV'
```

5) As you can see the genomic region selected for the SNAPPy's analysis is 'GAG-POL-ENV' which basically consists in the HIV-1 regions named GAG, POL, and ENV concatenated (according to the HXB2 (K03455) reference genome). Further on this tutorial we will use different genomic regions.

6) In order to perform the alignment of the sequences in the folder 'input' for the region specified in 'config.yaml' simple type:

```
snakemake align_all
```

Expected last line of the result:

```
5 of 5 steps (100%) done
```

7) As you can see SNAPPy started executing the tasks needed (at green) to obtain the requested multiple sequence alignment (MSA). Lets see which files and folders SNAPPy created:

```
ls
```

8) The folder 'aligned' and the file 'all_aligned.fasta' were created. The folder 'aligned' contains intermediate files used to create the final MSA. You can now use a text editor or your favorite FASTA file reader (for instance AliView) to look at the 'all_aligned.fasta' file. As you can see it contains a lot of gaps ('-') because the aligned sequences only contained information for the GAG region and we requested an alignment to the HXB2 reference genome for the GAG,POL, and ENV regions (as specified in the 'config.yaml file'). The produced sequences are of length 6918 nucleotides.

9) Now lets save the obtained alignment to a new folder called 'safe_outputs' and use a SNAPPy rule to clean all the outputs previously created:

```
mkdir safe_outputs
cp all_aligned.fasta  safe_outputs/msa_gag_pol_env.fasta
snakemake delete_all_outputs
```

10) The SNAPPy rule 'delete_all_outputs' is extremely useful to quickly delete files from previous runs but make sure that if you want to save outputs that you want to keep before running this rule (as we did above).

11) Lets modify the 'config.yaml' file to obtain an alignment only for the GAG region. Open The 'config.yaml' in your favorite text editor and edit it so it looks like this:

```
genomic_region:
  'GAG'
```

12) Now lets ask SNAPPy to align the sequences in the input folder:

```
snakemake align_all
```

13) The same outputs as before were created. If we now evaluate the 'all_aligned.fasta' file we can see it has far less gaps ('-'). As stated before the inputs sequences only contained information for the GAG region, and these outputs (of length 1503 nucleotides) only have that said region.

14) Fell free to test SNAPPy to create MSA for other HIV-1 sequences or using other genomic regions. Don't forget to always clean and save the outputs from previous runs if you want to keep them. If you are planning on exploring with different genomic regions don't forget to edit the 'config.yaml file'. The implemented genomic regions in SNAPPy are:

```
'GAG', 'PR', 'RT', 'PR-RT',
'INT', 'POL', 'ENV',
'GAG-POL-ENV'
```

15) That's it for this tutorial! Don't forget that if you plan on using SNAPPy for subtyping the 'config.yaml' file always needs to indicate the option 'GAG-POL-ENV'.

2.3 3: Result Analysis

In this tutorial we will give a more in-depth look at the outputs created by SNAPPy in the subtyping process. This tutorial starts after Tuturial *1: Subtyping with SNAPPy*, and uses the outputs created in that tutorial. If you have not run Tutorial 1 yet or no longer have its outputs in the folder please do so before the next steps.

1) Look at the content of the snappy folder:

```
ls
```

Expected result:

```
aligned  blast  config.yaml  data  environment.yaml  input  LICENSE.txt  README.md␣
↪scripts  Snakefile  snappy_installer_x86_64.sh  report_subtype_results.csv subtype_
↪results.csv  test  test_pipeline.py  trees
```

2) In this tutorial we will focus on the files 'report_subtype_results.csv' and 'subtype_results.csv'. To read and edit them fell free to use your favorite text editor or spreadsheet reader. Lets open the 'subtype_results.csv' file.

3) This is an extremely simple file with only tree columns: 'id', 'name', 'result'.

3.1) The 'id' field is only important if you want to evaluate by yourself the intermediate files created by SNAPPy in the 'aligned', 'blast' and 'trees' folders. For instance the files refering to the FASTA with the header 'test01' will be named '0' like: 'aligned/0.fasta', 'aligned/aligned_0.fasta', 'blast/blast_0.txt', 'blast/recblast_0.txt', 'trees/all_0.nwk', 'trees/pure_0.nwk', and 'trees/recomb_0.nwk'. If you want to know more on why and how those files were created please see the *How it works* section.

3.2) The 'name' field corresponds to the headers found in the HIV-1 sequences in the file inside the 'input' folder. This will be the field that allows the user to cross the SNAPPy outputs with the user nomenclature.

3.3) The 'result' field only contains the output produced by SNAPPy regathering that FASTA sequence subtype. No information is displayed regathering the analysis by BLAST or phylogenetic inference or how the decision was made. That information is in the file 'report_subtype_results.csv'.

4) Now lets open the report_subtype_results.csv. This output has 12 columns named: 'id', 'name', 'result', 'recomb_result', 'node_all_refs', 's_node_all_refs', 'node_pure_refs', 's_node_pure_refs', 'node_recomb_refs', 's_node_recomb_refs', 'closser_ref', and 'rule'. The first three are exactly the same as for the 'subtype_results.csv file' (explained it points 3.1 to 3.3). The remaining will be described in the following topics.

4.1) The field 'recomb_result' referes to an output obtained in a sliding with multiple BLASTs of the input. What this means is that the input was sliced multiple times and each slice served as a BLAST input to a database containing HIV-1 reference sequences. This test was mainly done looking for evidence of recombination in the target sequence. If you want to read more about the sliding window applied or the reference sequences used please read the *How it works* section.

4.2) The fields 'node_all_refs', 'node_pure_refs', and 'node_recomb_refs' correspond to the output of the phylogenetic inference using FastTree. These fields demonstrate if the target sequence was in a monophyletic clade with a group of HIV-1 reference sequences of only one subtype/circulation recombinant form (CRF). As the name indicate the 'node_all_refs' was inferred from a phylogenetic tree with potentially references from subtypes and CRFS, for the 'node_pure_refs' only subtype references were present, and for the 'node_recomb_refs' only CRFs references were present. If you want to know more about the parameters used in these phylogenetic inferences and/or the references used please go to the section *How it works*.

4.3) The fields 's_node_all_refs', 's_node_pure_refs', and 's_node_recomb_refs' contain the support values for the monophyletic nodes where the criteria explain in point 4.2 are meet. To obtain these support values the Shimodaira-Hasegawa test as implemented in FastTree was used.

4.4) The field 'closser_ref' shows the subtype or CRF of the reference sequence that showed to be closer to the target sequence in a BLAST analysis with all HIV-1 reference sequences used. If you want to know more about this BLAST or the reference sequences used please go to the section *How it works*.

4.5) The field 'rule' is merely informative and shows which SNAPPy 'rule' was used to make the decision about the subtyping output in the 'result' field based on the other fields. If you want to know more about these rules please go to the section *How it works*.

5) We believe that simplifying the 'subtype_results.csv' file allows users to quickly use SNAPPy, while providing the 'report_subtype_results.csv' file allows the user to observe the intermediate results created by SNAPPy and the decisions made. Please keep in mind that there will be cases harder to subtype that others, but you can always come back to the report_subtype_results.csv file and understand why SNAPPy outputted a given result.

6) After this tutorial we believe that you are equipped with the knowledge to use SNAPPy and completely understand its outputs.

CHAPTER 3

SNAPPy commands

This is a list of the implemented functionalities in SNAPPy:

- Basic subtyping:

```
snakemake subtype_all
```

- Subtyping using n cpu cores:

```
snakemake subtype_all --cores n
```

- Basic alignment:

```
snakemake align_all
```

- Alignment using n cpu cores:

```
snakemake align_all --cores n
```

- Clean the SNAPPy folder from previous runs outputs:

```
snakemake delete_all_outputs
```

- Clean the SNAPPy folder from previous runs intermediate files (but not outputs):

```
snakemake delete_interm_files
```

- Compress to a file and clean the SNAPPy folder from previous runs intermediate files (but not outputs):

```
snakemake compress_and_del_inter_files
```

- Compress everything inside the SNAPPy folder to a file:

```
snakemake compress_all
```

Note: Please note that SNAPPy is built on top of Snakemake, and therefore should retain all the functionalities of this pipeline software.

CHAPTER 4

How it works

4.1 Quick sum-up

As the name indicates SNAPPy is a pipeline, a series of tools used in succession or parallel to achieve an end. The general flow of the tool can be see in Fugure 1.

1) From a multiple sequence alignment a series of single FASTA sequence files are build.
2) Each of these FASTA files is then aligned to the HIV-1 reference genome (HXB2 (K03455)).
3) Each of the aligned files are them BLASted against a database of HIV-1 reference sequences (see *Reference Sequences*). The closest sequence according to the BLAST analysis is outputted in the 'report_subtype_results.csv' file in the column 'closser_ref'.
4) The BLAST results are them used to make 3 groups of references sequences of size 48: 1) all references; 2) only references of subtypes and sub-subtypes ; 3) only references of circulating recombination forms (CRF). - Each of these tree groups, together with the target sequence and an non-HIV-1 sequence (see *Reference Sequences*), are used to create 3 phylogenetic trees. These phylogenetic trees are then evaluated to see if the target sequence belong to a monophyletic clade with references of one, and only one, subtype or CRF. If this happens the subtype/CRF and the support values (Shimodaira-Hasegawa test) for the respective node are outputted in the 'report_subtype_results.csv' file in the columns 'node_all_refs', 's_node_all_refs', 'node_pure_refs', 's_node_pure_refs', 'node_recomb_refs', and 's_node_recomb_refs'.
5) In parallel a *Sliding window BLAST* is performed against a database of HIV-1 reference sequences (see *Reference Sequences*), in order to look for evidence of recombination. The output from the sliding window BLAST can be seen in the column 'recomb_result' in the 'report_subtype_results.csv' file.
6) These eight outputs are them combined and evaluated using a set of *Decision Rules* creating the final SNAPPy output, which can be seen in the column 'result' in both output files ('subtype_results.csv' and 'report_subtype_results.csv').
7) SNAPPy can also be used to perform intermediate tasks of its pipeline, such as multiple sequence alignment (MSA). This characteristic is extremely useful for large scale HIV-1 alignment against a reference genome (HXB2 (K03455)). The MSA alignment created by SNAPPy is outputted to the file 'all_aligned.fasta'.

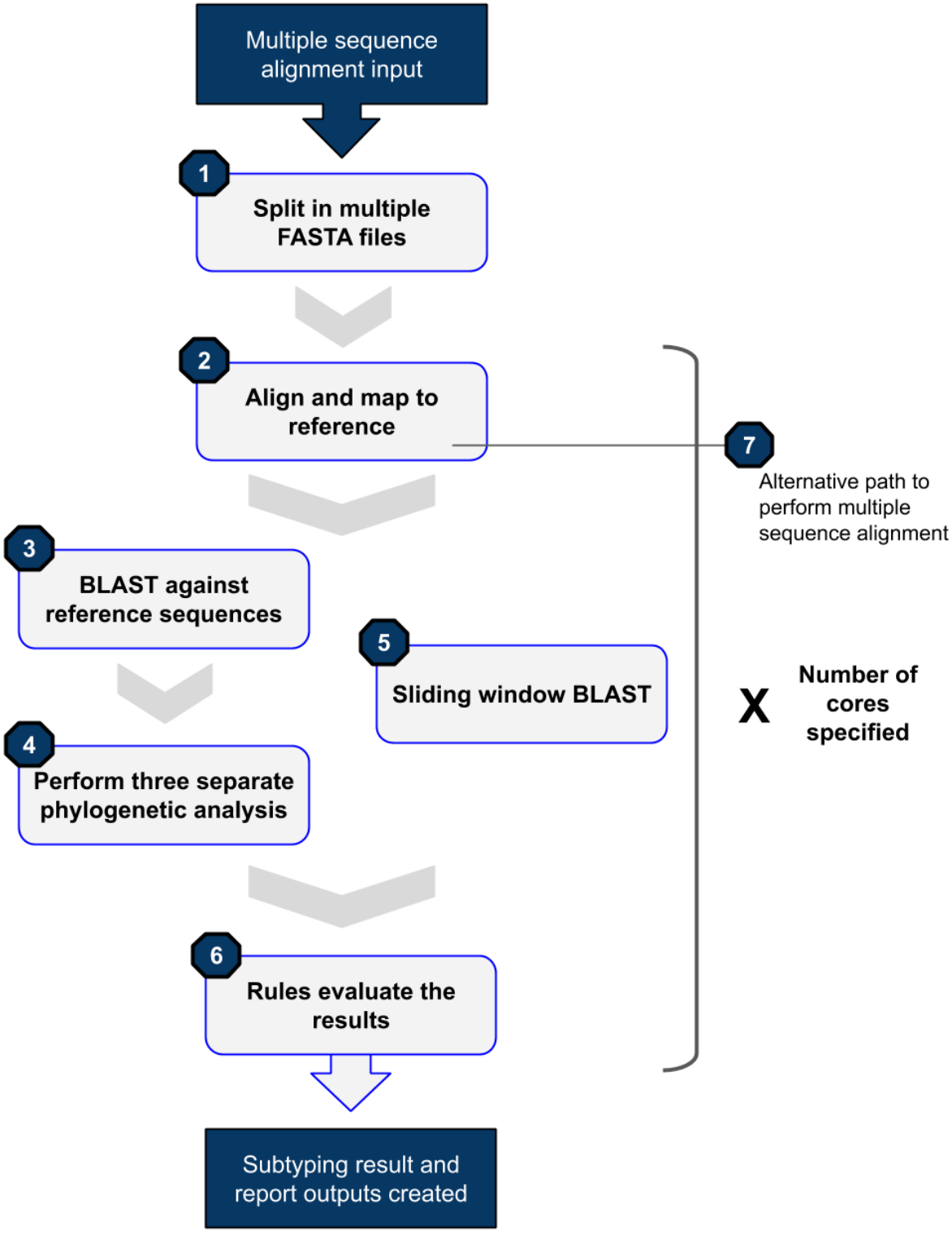

Fig. 1: **Figure 1: Schematic representation of the SNAPPy workflow.**

In both subtyping outputs ('subtype_results.csv' and 'report_subtype_results.csv') there is also a column named 'id' which refers to the internal identifier used inside SNAPPy instead of the FASTA file header. If one wants to check the SNAPPys intermediate files first look for the connection between the 'id' that correspond to the desired FASTA in the 'report_subtype_results.csv' file.

If you desire to know more about how each of these steps are processed inside SNAPPy please read the following sections. If you want to use SNAPPy here is a list of the *SNAPPy commands*.

4.2 Reference Sequences

In all instances of SNAPPy the HXB2 (K03455) reference genome was used as a genomic position map reference, the HIV sequence database as a great visualization of this reference genome .

In the phylogenetic inferences the CONSENSUS_CPZ sequence from the HIV sequence database was used as outgroup for tree rooting (Alignment type: Consensus/Ancestral, Year: 2002, Organism: Other SIV, DNA/Protein: DNA, Region: genome, Subtype: ALL, Format: FASTA, Alignment ID : S02CG1).

We tried our best to create the best possible dataset of HIV-1 reference sequences. This task was extremely challenging. We based our dataset on previously curated data like the HIV sequence database 2010 subtype reference genomes sequences compendium and the HIV Drug Resistance Database reference sequences . Our goal was to create a comphresensive yet not too extensive dataset of HIV-1 references. The subtype reference dataset used in SNAPPy at the moment consists of 493 genomic sequences, and can be downloaded here. following there is a list of the ids of these 493 sequences and the subtype they were used as a reference of:

A1-DQ676872, A1-AB253421, A1-AB253429, A2-AF286238, A2-GU201516, A2-AF286237, B-K03455, B-AY423387, B-AY173951, B-AY331295, C-U52953, C-U46016, C-AF067155, C-AY772699, D-K03454, D-AY371157, D-AY253311, D-U88824, F1-AF077336, F1-AF005494, F1-AF075703, F1-AJ249238, F2-AY371158, F2-AJ249236, F2-AJ249237, F2-AF377956, G-AF084936, G-AF061641, G-U88826, G-AY612637, H-AF190127, H-AF190128, H-AF005496, H-FJ711703, J-EF614151, J-GU237072, J-AF082394, K-AJ249235, K-AJ249239, 01_AE-GQ477441, 01_AE-GU564221, 01_AE-U54771, 02_AG-AY271690, 02_AG-AB485636, 02_AG-L39106, 03_AB-AF193276, 04_cpx-AF049337, 04_cpx-AF119820, 04_cpx-AF119819, 05_DF-AF076998, 05_DF-AF193253, 05_DF-AY227107, 06_cpx-AF064699, 06_cpx-AY535659, 06_cpx-AB286851, 07_BC-EF368372, 07_BC-EF368370, 07_BC-AF286230, 08_BC-HM067748, 08_BC-AY008715, 09_cpx-AJ866553, 09_cpx-AY093605, 09_cpx-AY093603, 09_cpx-AY093607, 10_CD-AF289548, 10_CD-AF289549, 10_CD-AF289550, 11_cpx-AF492624, 11_cpx-AF492623, 11_cpx-AJ291718, 12_BF-AF408629, 12_BF-AF408630, 12_BF-AF385936, 13_cpx-DQ845388, 13_cpx-DQ845387, 13_cpx-AF460972, 14_BG-AF450096, 14_BG-AF450097, 14_BG-GU230137, 15_01B-DQ354120, 15_01B-AF516184, 15_01B-AF530576, 16_A2D-AY945736, 16_A2D-AF286239, 17_BF-EU581825, 17_BF-EU581827, 17_BF-EU581828, 18_cpx-AF377959, 18_cpx-AY586541, 18_cpx-AY894993, 19_cpx-AY588971, 19_cpx-AY588970, 19_cpx-AY894994, 20_BG-AY586545, 21_A2D-AY945737, 21_A2D-AF457051, 21_A2D-AF457072, 22_01A1-AY371159, 22_01A1-GQ229529, 23_BG-AY900571, 23_BG-AY900572, 24_BG-AY900574, 24_BG-AY900575, 24_BG-FJ670526, 25_cpx-EU693240, 25_cpx-EU697906, 25_cpx-EU697908, 26_A5U-FM877780, 26_A5U-FM877782, 26_A5U-FM877777, 27_cpx-AJ404325, 27_cpx-AM851091, 28_BF-DQ085872, 28_BF-DQ085873, 28_BF-DQ085874, 29_BF-DQ085876, 29_BF-AY771590, 29_BF-DQ085871, 31_BC-EF091932, 31_BC-AY727526, 31_BC-AY727527, 32_06A6-AY535660, 33_01B-AB547464, 33_01B-DQ366659, 33_01B-DQ366662, 34_01B-EF165541, 35_AD-EF158043, 35_AD-EF158040, 35_AD-EF158041, 36_cpx-EF087995, 36_cpx-EF087994, 37_cpx-EF116594, 37_cpx-AF377957, 38_BF-FJ213781, 38_BF-FJ213782, 38_BF-FJ213780, 39_BF-EU735534, 39_BF-EU735536, 39_BF-EU735535, 40_BF-EU735538, 40_BF-EU735540, 40_BF-EU735539, 42_BF-EU170155, 43_02G-EU697904, 43_02G-EU697907, 43_02G-EU697909, 44_BF-FJ358521, 45_cpx-FN392874, 45_cpx-FN392876, 45_cpx-FN392877, 46_BF-DQ358801, 46_BF-DQ358802, 46_BF-HM026456, 47_BF-GQ372987, 47_BF-FJ670529, 49_cpx-HQ385477, 49_cpx-HQ385479, 49_cpx-HQ385478, O-L20587, O-L20571, O-AY169812, O-AJ302647, N-AY532635, N-AJ006022, N-AJ271370, P-HQ179987, P-GU111555, 42_BF-EU170142, 42_BF-EU170151, 42_BF-EU170147, 03_AB-AF193277, 03_AB-AF414006, 44_BF-AY536235, 20_BG-AY586544, 20_BG-DQ020274, 32_06A6-DQ167215, 34_01B-EF165539, 34_01B-EF165540, 48_01B-GQ175881, 48_01B-GQ175882,

48_01B-GQ175883, 32_06A6-KM606632, 20_BG-KT276270, 41_CD-KX907411, 41_CD-KX907417, 41_CD-KX907430, 03_AB-MF109476, 50_A1D-JN417241, 51_01B-JN029801, 52_01B-AY945734, 53_01B-JX390610, 54_01B-JX390977, 55_01B-JX574663, 56_cpx-KC852174, 57_BC-KC899008, 58_01B-KC522031, 59_01B-KC462190, 60_BC-KC899079, 61_BC-KC990124, 62_BC-KC870037, 63_02A-JX500706, 64_BC-KC870043, 65_cpx-KC870030, 67_01B-KC183779, 68_01B-KC183782, 69_01B-LC027100, 70_BF-KJ849809, 70_BF-KJ849761, 71_BF-DQ358811, 71_BF-KJ849775, 72_BF-KJ671533, 72_BF-KJ671537, 73_BG-KM248765, 74_01B-KR019770, 78_cpx-KU161143, 53_01B-JX390611, 53_01B-JX390612, 63_02A-KJ197201, 77_cpx-KX673818, 52_01B-DQ366664, 54_01B-JX390976, 57_BC-KC870044, 59_01B-KJ484433, 60_BC-KC899081, 62_BC-KC870034, 67_01B-KC183780, 69_01B-AB845349, 74_01B-KR019771, 78_cpx-KU161145, 50_A1D-JN417240, 51_01B-KJ485697, 55_01B-KF927150, 56_cpx-KC852173, 58_01B-KC522033, 61_BC-KC990126, 64_BC-KC898994, 65_cpx-MH051841, 68_01B-KF758551, 73_BG-AY882421, 77_cpx-KX673820, 63_02A-JX500705, 63_02A-JX500700, 63_02A-JX500699, 63_02A-JN230353, 63_02A-KJ197200, 63_02A-KJ197202, 63_02A-JX500704, 70_BF-KJ849758, 70_BF-KJ849762, 71_BF-KJ849760, 71_BF-KJ849769, 71_BF-KJ849771, 71_BF-KJ849777, 71_BF-KJ849778, 71_BF-KT427816, 72_BF-KJ671534, 72_BF-KJ671535, 72_BF-KJ671536, 79_0107-KY216146, 79_0107-KY216147, 79_0107-KY216148, 82_cpx-KU820825, 82_cpx-KU820831, 82_cpx-KU820836, 82_cpx-KU820837, 82_cpx-KU820844, 82_cpx-KU820845, 83_cpx-KU820823, 83_cpx-KU820824, 83_cpx-KU820826, 83_cpx-KU820828, 83_cpx-KU820829, 83_cpx-KU820833, 83_cpx-KU820834, 83_cpx-KU820839, 83_cpx-KU820842, 83_cpx-KU820843, 83_cpx-KU820847, 85_BC-KU992928, 85_BC-KU992929, 85_BC-KU992930, 85_BC-KU992931, 85_BC-KU992932, 85_BC-KU992934, 85_BC-KU992935, 85_BC-KU992936, 85_BC-KU992937, 86_BC-KX582251, 86_BC-KX582250, 86_BC-KX582249, 87_cpx-KC899012, 87_cpx-KC898992, 87_cpx-KF250408, 88_BC-KC898979, 88_BC-KC898975, 88_BC-KF250402, 90_BF1-KY628225, 90_BF1-KY628224, 90_BF1-KY628223, 90_BF1-KY628222, 90_BF1-KY628221, 90_BF1-KY628220, 90_BF1-KY628219, 90_BF1-KY628218, 90_BF1-KY628217, 90_BF1-KY628216, 90_BF1-KY628215, 92_C2U-MF372652, 92_C2U-MF372648, 92_C2U-MF372647, 92_C2U-MF372645, 93_cpx-MF372651, 93_cpx-MF372649, 93_cpx-MF372646, 94_cpx-MH141491, 94_cpx-MH141492, 94_cpx-MH141493, 94_cpx-MH141494, 96_cpx-KF850149, 96_cpx-MG518477, 96_cpx-MG518476, 80_0107-MH843712, 80_0107-MH843713, A1-AF069670, A1-M62320, A1-AF484509, A1-EF545108, C-AF443091, H-KU310618, 11_cpx-AY371152, 11_cpx-AJ291720, 16_A2D-AF457060, 22_01A1-GQ229530, 25_cpx-DQ826726, 26_A5U-FM877778, 44_BF-AY536238, 45_cpx-FN392875, 01_AE-KP411840, 01_AE-KP411841, 01_AE-KP718930, 02_AG-AB052867, 02_AG-AJ239083, 02_AG-AJ508595, 02_AG-AY444811, 02_AG-FJ388896, 02_AG-FJ694791, 02_AG-JF683786, 02_AG-JF683795, 02_AG-JN248585, 02_AG-KU749413, 02_AG-KM606636, 02_AG-KP411843, 02_AG-KP411844, 02_AG-KR067668, 02_AG-KT124792, 06_cpx-KU168301, 06_cpx-KX389609, 09_cpx-AJ866556, 11_cpx-AJ291719, 11_cpx-AY371151, 11_cpx-JF683802, 11_cpx-KX389633, 18_cpx-AY371166, 19_cpx-AY894995, 25_cpx-KY392772, A1-AF069669, A1-AF069671, A1-AF457068, A1-AY521629, A1-AB287378, A1-DQ396400, A1-AM000053, A1-EU861977, A1-FJ388893, A1-FJ670519, A1-KU749409, A1-KT152839, A1-KX232613, B-AF362994, B-AF005495, B-AY795904, B-AY682547, B-DQ207940, B-FJ460499, B-DQ358806, B-DQ383750, B-DQ383752, B-AB221125, B-DQ396398, B-EF637049, B-EF637050, B-EF637054, B-EF637056, B-FJ195091, B-FJ388890, B-HM030559, B-GQ372988, B-EU839596, B-EU839600, B-EU839606, B-JF320008, B-JF320144, B-JF683738, B-JF683796, B-JF683801, B-HQ215554, B-JN235958, B-JN251896, B-JN692432, B-JN692433, B-JN692439, B-JN692448, B-JN692455, B-JN692460, B-JN692467, B-JF804810, B-JF804812, B-JF804813, B-KY658689, B-KF384810, B-KJ849785, B-KJ849804, B-KJ849808, B-KJ849814, B-KJ849780, B-KP411822, B-KY465969, B-KT427650, B-KT427671, B-KT427681, B-KT427707, B-KT427720, B-KT427735, B-KT427746, B-KT427757, B-KT427811, B-KT427828, B-KT124761, C-AF286223, C-AF286227, C-AF286233, C-AF286234, C-AF457061, C-AY255823, C-AY255824, C-AY255825, C-DQ207941, C-AY734554, C-AB286849, C-EU786673, C-FJ388901, C-GQ999983, C-KU749412, C-KU749430, C-KP411830, C-KP411834, C-KY496624, C-KT022371, C-KT124786, C-KU319528, C-KU319529, C-KU319551, C-KX907353, C-KY392767, D-AF442569, D-AF484514, D-AY773338, D-AY795907, D-AY444799, D-DQ054367, D-DQ912823, D-FJ388945, D-KU749394, D-KU168272, D-KX907406, D-KY392769, F1-DQ189088, F1-FJ900266, F1-AB485658, F1-GQ290462, F1-JX140671, F1-KY392770, F2-JX140672, G-DQ168573, G-AB231893, G-AY586547, G-JN106043, G-KJ948662, G-KU168277, H-KU168273, H-KY392777, J-KU168280, J-KU310620

4.3 Alignment to reference

After splitting the MSA in several FASTA files each of them is aligned to the HIV-1 reference genome (HXB2 (K03455)). The module SeqIO from Biopython is used to parse and manipulate the FASTA files in SNAPPy. The alignment is done using MAFFT v7.245. The alignment method used does not allow any gaps in the reference sequence (comand: 'mafft –quiet –add "target" –keeplength "reference"). After the alignment is performed the target sequence is trimed to only contain the genomic region specified by the user in the 'config.yaml' file. Being the currently available options 'GAG', 'PR', 'RT', 'PR-RT', 'INT', 'POL', 'ENV', and 'GAG-POL-ENV'. The resulting file is them written to the folder 'aligned' with the following notation: aligned_{internal_id}.fasta.

4.4 BLAST

The curated and aligned sequence files obtained are them BLASTed against a local database of HIV-1 reference sequences (see *Reference Sequences*). For this task BLAST v2.7.1 is used (comand; 'blastn -db "references" -query "target" -out "blast_output" -word_size 10 -outfmt 10 -evalue 1.e-10'). The BLAST parameters 'word_size' = 10 and "evalue" cutoff = 1.e-10 showed good and consistent performance for the case of highly similar sequences. The results were sorted by bitscore (higher is better). The BLAST result with the best score is outputted in the 'report_subtype_results.csv' file in the column 'closser_ref'. The BLAST results are also used to make 3 groups of references sequences: containing the first 48 results; containing the first 48 results of only subtype references; containing the first 48 results of only CRF references. These three groups of reference sequences are then used in the phylogenetic analysis (see *Phylogenetic inference*. The intermediate files of the BLAST analysis are outputted to the folder 'blast' with the following notation: blast_{internal_id}.txt. Please notice that for the split in subtype and CRF references in this step of SNAPPy CRFs 01 and 02 are treated as subtypes an not CRFs. These decision was made based on the high prevalence of this CRFs and their ambiguous origin (Gao F, et al. J Virol. 1996; Abecasis AB, el al. J. Virol. 2007)

4.5 Phylogenetic inference

The target sequence (see *Alignment to reference*) and a non-HIV-1 sequence (for rooting, see *Reference Sequences*) are added to each of the three previously selected groups of 48 references (see *BLAST*). Groups of 50 sequences showed to be a contained and yet comprehensive set of sequences to perform the phylogenetic inference. To perform the phylogenetic analysis FastTree v2.1.10 was used (comand: fasttree -quiet -gtr -nopr -nt "msa_50_seqs"'). The Biopython module Phylo is used to parse and manipulate the phylogenetic trees created in SNAPPy. After rooting on the outgroup it is inferred if the target sequence is in a monophyletic clade with references of a same subtype/CRF. If that happens the subtype/CRF of those references is outputted together with the support values for that node (Shimodaira-Hasegawa test, as implemented in FastTree). Resulting in six output columns in the 'report_subtype_results.csv' file: 'node_all_refs', 's_node_all_refs', 'node_pure_refs', 's_node_pure_refs', 'node_recomb_refs', and 's_node_recomb_refs'. The intermediate files of the phylogenetic analysis are outputted to the folder 'trees' with the following notation: {type}_{internal_id}.nwk (being {type} 'all', 'pure', and 'recomb' referring to the set of references used for that phylogenetic reconstruction).

4.6 Sliding window BLAST

This step of the pipeline starts from the sequence files created in the *Alignment to reference* section. The positions with gaps ('-') in the target sequence are excluded. The length of the sliding window is 400 nucleotides. Fragments with length inferior to 400 will not be processed by this approach, and 'impossible to test recomb (lenght < 400bp)' will be written to the output file. The step size is 50 nucleotides, which creates 8 bins for each window. The result for each BLAST window is the subtype of the top result (bitscore). If more than one sequence of different subtypes

have the same top score the output for all bins of that window is null. If the BLAST fails or no output is produced the output for all bins of that window is null. At the end of all sliding windows being processes several bins may have multiple outputs, a majority rule is applied to decide the final subtype for that bin. In case of tie the result for that bin is null. In this sliding window BLAST we are only BLASting against a database of HIV-1 subtype references, plus the CRFs 01 and 02 due to the reasons previously discussed in *BLAST*. The BLAST command used for each window is : 'blastn -db "database" -query "target -out', "output_file_name" -word_size 10 -outfmt 10 -evalue 1.e-100'. The database file for the BLAST can be consulted in this link . Similarly to what was mentioned in *BLAST* section these parameters proved to be very useful in the case of highly similar sequences like HIV-1. Please consult Figure 2 for a graphic explanation of this process. The resulting file is them written to the folder 'blast' with the following notation: recblast_{internal_id}.txt.

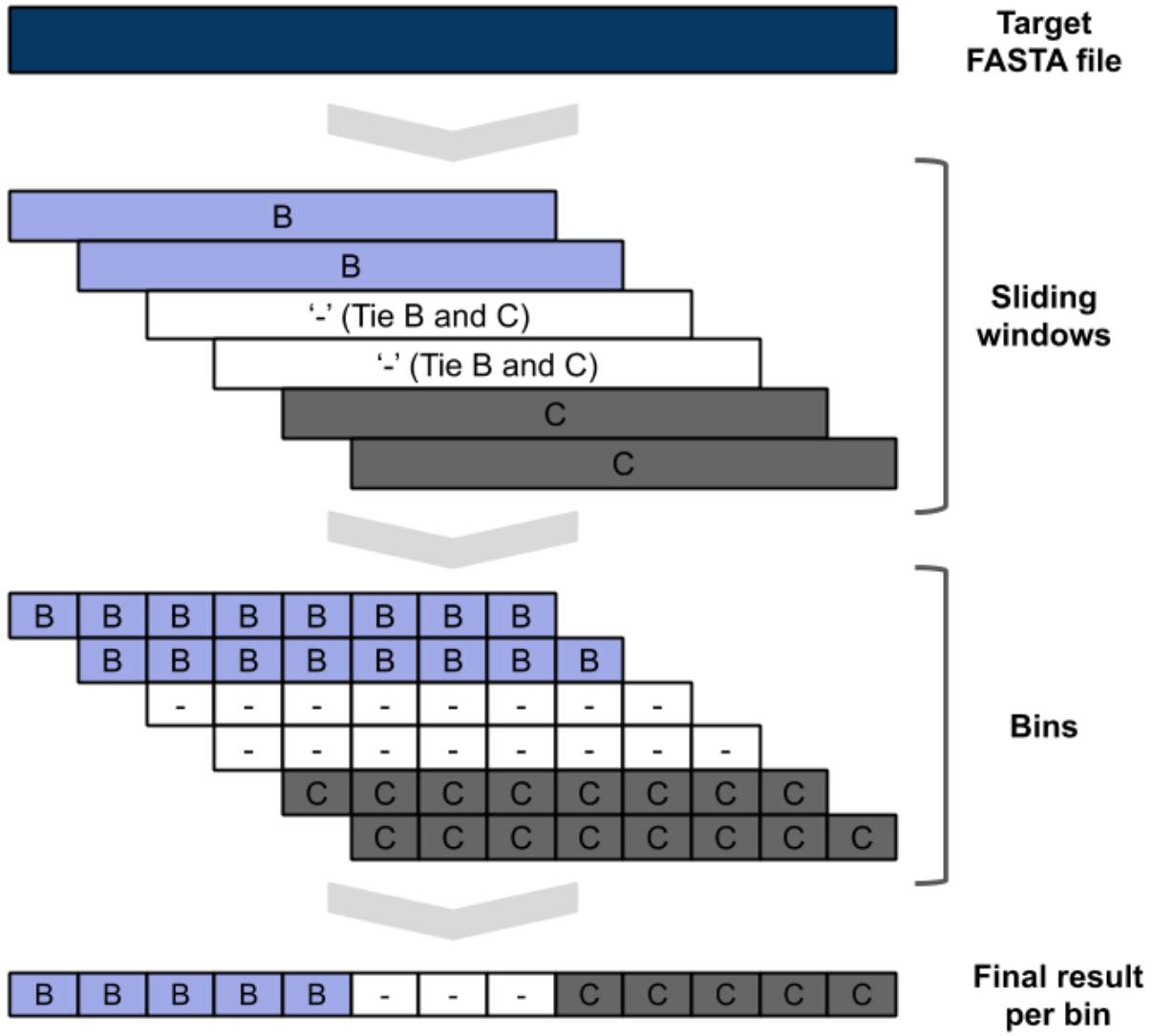

Fig. 2: **Figure 2: Schematic representation of the sliding window approach. Demonstrating evidence of B/C recombination.**

4.7 Decision Rules

Given the previously described outputs produced by SNAPPY a set of decision rules were built. The rules are executed in the order they are shown. This means that for instance rule p2 is only applied after the requirements were not meet for rules p1 and c1. To simplify the rules description we describe the results from the Sliding window BLAST as follows: no recombination - if only one subtype/CRF is outputted; simple recombination - if two subtypes/CRFs are outputted, complex recombination - if more than two subtypes/CRFs are outputted. Tree 'all', 'pure', and 'recomb' refer, respectively, to the results from the phylogenies with all closest reference sequences, only subtype closest reference sequences, and only CRF closest reference sequences. Good support refers to Shimodaira-Hasegawa test (as implemented in FasTree) result >= 0.7, while a great support refers to results >= 0.9. This nomenclature has as only objective simplifying the description of the following rules and should not be applied outside this context:

Rules:

- p1: All analysis report the same output. Output: all tree result;
- c1: Simple recombination and all tree; recomb tree; and closer reference are equal. Both trees support is good. Output: all tree result;
- p2: All tree; pure tree; and recomb result are equal. Both trees support is good. Output: all tree result;
- p3: All tree; pure tree; and closer reference are equal. Both trees support is good. Output: all tree result;
- c2: All tree; recomb tree; and closer reference are equal. Both trees support is good. Output: all tree result;
- p4: Pure tree; closer reference; and recomb result are equal. Tree support is great. Output: pure tree result;
- c3: Recomb tree; closer reference; and recomb is simple. Tree support is great. Output: recomb tree result;
- b1: All tree; closer reference; and recomb result are equal. Tree support is great. Output: all tree result;
- **There is evidence of complex recombination:**
  - c4: All tree and recomb tree give same result which is a CRF. Both trees support is good. Output: all tree result;
  - p5: All tree; pure tree; and closer reference are equal. At least one of the trees has great support. Output: all tree result;
  - c5: All tree; recomb tree; and closer reference give same result which is a CRF. Output: all tree result;
  - p6: All tree; pure tree; and closer reference are equal. Output: all tree result;
  - u1: The remaining cases with evidence of complex recombination. Output: 'URF_CPX';
- **There is evidence of simple recombination:**
  - c6: All tree and recomb tree give same result which is a CRF. Both trees support is good. Output: all tree result;
  - p7: All tree; pure tree; and closer reference are equal. At least one of the trees has great support. Output: all tree result;
  - c7: All tree; recomb tree; and closer reference give same result which is a CRF. Output: all tree result;
  - p8: All tree; pure tree; and closer reference are equal. Output: all tree result;
  - u2: Remaining cases. Output: URF between the two results of the sliding window blast result;
- p9: All tree equal pure tree. Both trees support is good. Output: all tree result;
- **If there is missing data regarding the recombination test:**
  - f1: If there is no result in the sliding window blast. Output: closer reference result;

  - f2: If both the closer reference and sliding window blast results are missing but all tree agrees with pure tree result: Output: all tree result;
  - f3: If both the closer reference and sliding window blast results are missing but all tree agrees with recomb tree result. Output: all tree result;
  - f4: Remaining cases. Output: 'impossible_to_determine';
- f5: There is no evidence or recombination and the sliding window blast result is not null: Output sliding window blast result.

4.8 Other

Some intermediate files produced by SNAPPy are deleted before the end of the process. This was done to avoid waisting unnecessary disk space and at the same time simplify the user experience. However, all the intermediate files created by SNAPPy that may contain useful information regarding the analysis and the decisions made by the pipeline are kept.

Please keep in mind that SNAPPy does several analysis and some of them produce a large amount of outputs. If you are using SNAPPy for large scale analysis please understand that a large portion of disk space may be needed. This is a tradeof between transparency and computational resources that we thought may be the best for the user. At the end of each SNAPPy run it will delete an snakemake hidden folder named '.snakemake' that occupies substantial amount of space. However, this folder contains all the logs about the tasks performed and may be useful for debugging. To change that behavior two lines need to be commented out in the 'Snakefile' ('onsuccess:' 'shutil.rmtree(".snakemake")'), if you decide to do so do it at your own risk.

CHAPTER 5

FAQ

Q: "Why should I use SNAPPy?

A: Do you want to do HIV-1 subtyping/alignment in large numbers? Do you have to process your HIV-1 sequences locally because ethic concerns or data privacy policy? Do you want a tool for HIV-1 subtyping/alignment that scales with the amount or computational resources you have? Do you want a tool that gives simple outputs but also allows you to dive deep in intermediate files and how the decisions were made? If you answered was yes to any or all or the previous questions SNAPPy is a tool for you.

Q: "Can SNAPPy run on a windows machine?

A: In theory it is possible to run SNAPPy on a windows machine using the Windows Subsystem for Linux , however at the moment we do not provide any support for this and if you decide to do so do it at your own risk. The main reason SNAPPy was build for Linux based systems is that it has dependencies that only work on these operating systems. We are considering options to make SNAPPy operative systems agnostic in future releases.

Q: "Is there any graphical user interface to use SNAPPy?

A: Currently, SNAPPy is a command base tool. However, we are trying to implement more user friendly ways to use SNAPPy without losing functionality.

Q: "Why using BLAST in SNAPPy?

A: We decided to use BLAST for the first version of SNAPPy because it is reliable, extensively tested and easy to implement. We are aware of more "modern" BLAST-like tools, but one needs to recognize that BLAST has decades of being "battle tested". This decision does not restrict the usage of other tools/software in future versions of SNAPPy.

Q: "Why using FastTree in SNAPPy?

A: After testing several phylogenetic tools we acknowledged that FasTree had the best ratio between phylogenetic inference quality and computational time. Moreover, FastTree has all the characteristics needed for the phylogenetic inference software for this version of SNAPPy. This decision does not restrict the usage of other tools/software in future versions of SNAPPy.

Q: "I used SNAPPy to subtype 100k sequences and the outputs are occupying a lot of disk space (several dozens gigabytes), is it normal?

A: Please keep in mind that SNAPPy is a complex pipeline and performs several analysis. Each of those generate outputs. Everything is multiplied by the number of inputs, even if each input occupies an almost negligible amount of

disk space the sum of all of them will result in a large value. At the end of each run SNAPPy automatically deletes an hidden folder created by Snakemake named '.snakemake', this folder is extremely large and its deletion will save some space (do not delete it if the pipeline is still being executed).

Q: "Why can I not put several FASTA files inside the input folder?

A: This was a development decision. This may change in the future but for the current SNAPPy version we believe that is the best option for both the user and the pipeline.

CHAPTER 6

How to Cite

text

CHAPTER 7

License

For aditional information please contact: INFO (MAIL)