# **NOTE: balded parts are meant to be replaced and unbolded once the order of figures and references is in the final report is established**

# PhyloWGS:

## Background survey

The methods of reconstructing a tumour phylogeny mainly fall into two camps: one is based on the tumour simple somatic mutation (SSM) data; others focus on the observed copy number variations (CNVs) of these SSMs. Taken seperately, these approaches comes with their limitations and varying levels of difficulty, either by making several consequential assumptions about the underlying nature of the evolution of the tumour (SSM-based approaches) or are severely limited to correctly infering only relatively simple tumour philogenies (CNV-based approaches). These approaches are generally limited to a certain degree of accuracy and it is even possible to demonstrate situations where using both of these approaches seperately on the same tumour sample yields incorrect results [**PHYLO**].

PhiloWGS [**PHYLO**] is a first method that falls in between of these two camps - it incorporates both the information about the tumour SSMs and their corresponding CNVs, while making fewer assumptions about the tumour philogeny along the way. With important considerations of the order of occurance between SSMs and CNVs, PhyloWGS accomplishes such a task via a complicated process that involves creating a pseudo-simple somatic mutations for each of the copy number variations and assigning all the affected SSMs within the region to these newly created pseudo-SSMs.

Based on Metropolis-Hastings algorithm that samples from a posterior distribution of a generative probabilistic model with non-parametric Bayesian priors, PhyloWGS also does not report only a single phylogenetic tree, but a tree for every converged Markov chain Monte Carlo simulation.

Written in Python and C++, PhyloWGS reports an increased performace over PyClone due to usage of powerful scientific C++ library GSL [**GSL**] and incorporation of CNV and SSM. From a software engineering perspective, it might be best described as an end-to-end piece of software, bundled with input parsers for a number of popular data formats for this type of task (such as Battenberg and TITAN) with a framework to implement your own (which we make use of) and a web-based locally hosted interface for interpreting results.

While the full scope of functionality that PhyloWGS offers is broader that what we need in constructing our data pipeline, we were able to pick parts of it that we made use of, despite having to make a few compromises along the way, in a method that will be descibed in the chapters that follow.

## Requirements

In order to use PhyloWGS seperately, we need the following tools and libraries. While the requirements in the official documentation are rather parsimonious, we will include the major version numbers we used in the pipeline below, in oder to achieve maximum reproducability:

1. Python 2.x interpreter (major version number 2.7)
2. Python 2.x version of NumPy (major version number 1.11)
3. Python 2.x version of SciPy (major version number 1.1)
4. Python 2.x version of ETE2 (major version number 2.3)
5. Python 2.x version of PyVCF (major version number 0.6)
6. GSL – GNU Scientific Library for C and C++ (major version number 2.5)
7. Python 2.x version of pandas (major version number 0.23)
8. getopt, the C library for parsing command line arguments in shell scripts (from the util-linux package, major version number 2.31)

While it is not explicitly mentioned, and a compiled version of GSL for Windows exist as part of Cygwin for Windows, for the ease of use we will assume the user uses a Unix-based operating system, such as Ubuntu Linux or MacOS.

In addition to downloading these tools, a PhyloWGS GitHub repository [**GITHUB**] must be cloned, navigated to, and the following line must be executed in the terminal:

g++ -o mh.o -O3 mh.cpp util.cpp `gsl-config --cflags --libs`

Figure 1: Compilation instructions for the C++ file

This ensures that the C++ file that uses GSL within PhyloWGS is compiled and the tool is ready to be used.

Note, that because of the provided framework within the source files to implement your own data parser and the differences of our input data versus what PhyloWGS normally expects, we have modified the original PhyloWGS source code in order to comply with our data, instead of writing our own external parsers for the PhyloWGS native parsers; a solution we judged to be the best at the time. Therefore, we bundled our data pipeline with our own modified version of PhyloWGS and do not recommend it to clone it seperately as it won’t work with our data natively.

In addition to the native requirements of PhyloWGS, we used number of other libraries in the PhyloWGS-related part of the pipeline, all convieniently part of the standart Python library, with a notable exception of pandas. As part of the pipeline, we had to convert widely different output formats of the three tools into a single, common one. We made use of time-saving and more elegant pandas DataFrame management capabilities to neatly save our transformed data into the common format files.

## Design and implementation

As previously mentioned, PhyloWGS is an end-to-end framework more than a library, and its native usage reflects that. In order to use PhyloWGS alone, a user must write a script that pre-parses the copy number variation (CNV) data (should a user wish so, as this is optional) for the main parser, which generates the inputs for the main scripts.

After executing the main script (either by running a single or multiple concurrent MCMC chains), a user must use another script to uncompress the resulting trees file, restructuring the data into now three seperate re-compressed .json files and move it to the result-viewing folder.

Lastly, a user must uncompress the three files again, launch the data indexing script, launch a HTTP server, open a browser and interpret the results, as visible **in the Figure 2**.

None of the interim scripts output simple text files with relevant data, none of them offer to disable the compression, and some of the most important metrics for our results (such as cancer cellular frequency (CCF) of a cluster, or determining the ‘best’ phylogenetic tree) is only done by front-end-supplying JavaScript files, executed only when the user navigates to the locally hosted interface to interpret the results.

After the initial impressions, we chose rather focus on fast conversion of one of the iterim results (to the common format), than the lengthly proccess of untangling and making changes to each and every script to get the output that we want.

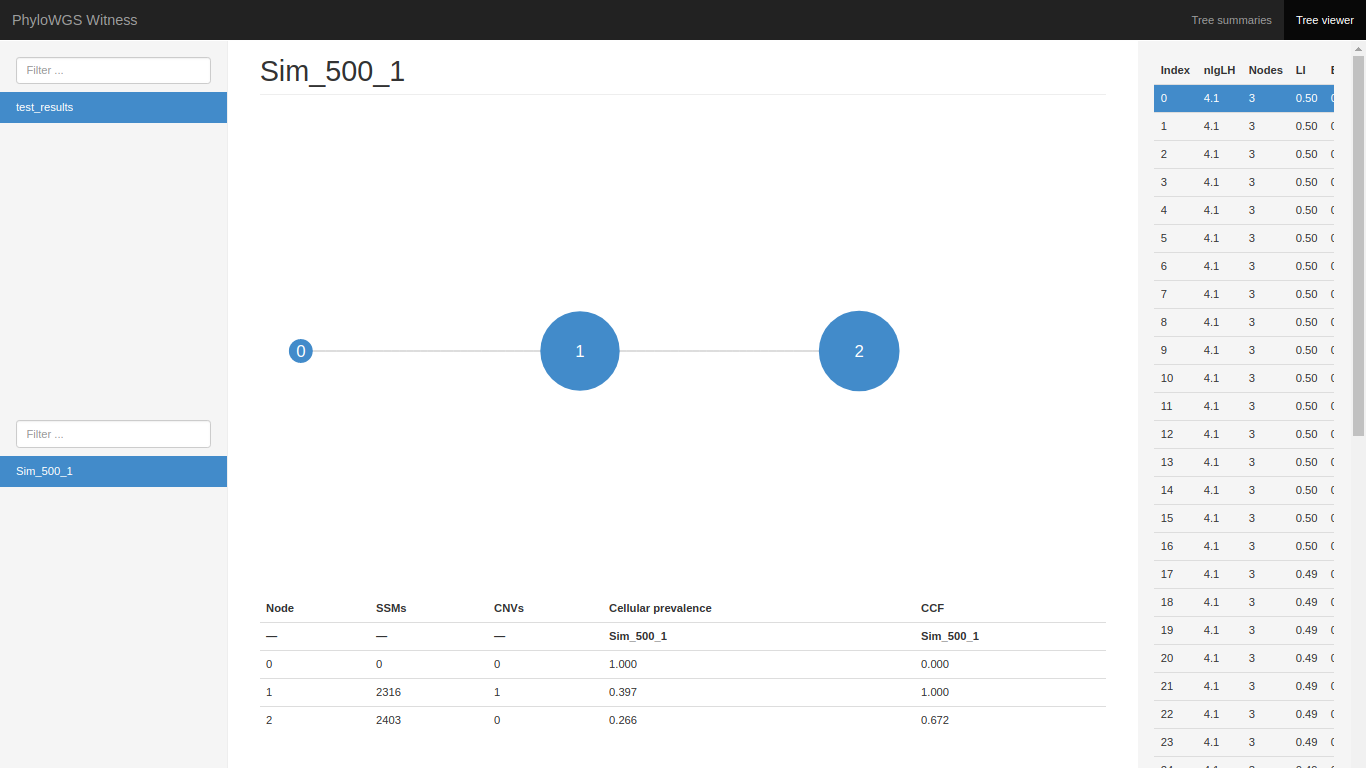


Figure 2: PhyloWGS native web-based interface for interpreting the results (for sample ‘Sample\_500\_11’), Tree Viewer tab

Therefore, in order to adapt PhyloWGS efficiently to the main pipeline, we had to complete three main software engineering tasks:

1. Supplied with the framework, create a custom pre-parser for the copy number variation data and a custom main parser for the raw .VCF file inputs and the output of the former to create the inputs for the main PhyloWGS pipeline: files ssm\_data.txt and cnv\_data.txt.
2. Create a shell script executing all the relevant native PhyloWGS scripts up to an acceptable level of the output. Execute the shell script from inside the pipeline class with the relevant parameters passed in each loop.
3. Devise a scalable, vectorized post-proccessing logic to extract and calculate the needed statistics in order to transform the compressed .json files into the common format text files.

Remarkably, we managed to achieve the running times of data post-processing logic only as slow as 0.14 seconds per sample, achieving the combined running time close to (or surpassing) PyClone when run with comparable parameters.

As with any tool that aims to reconstruct the tumour phylogeny, PhyloWGS can be customized via a plethora of parameters. However, in our data pipeline, we exposed the user only to the common ones that we judged are most likely to be changed:

1. Number of MCMC chains to be run concurrently;
2. Number of MCMC samples for each chain;
3. Number of burn-in samples for each chain;
4. Number of Metropolis-Hastings iterations;

Lastly, there are a couple of very important distinctions need to be made at his point. As may be noticable **by Figure 2:**

1. The native way of choosing the ‘best’ phylogenetic tree is different in our pipeline and in the source code of PhyloWGS. We choose the best tree as the most likely one, in other words, the one with the highest reported likelihood. In the PhyloWGS source code, the ‘best’ tree is considered the the tree that is most similar to the other trees [**BESTTREE**]
2. Some of the clusters might have cellular prevalence and cancer cell fraction numbers approaching the value of 1. This is because of there are two different ways to define cellular prevalence, and a user might be puzzled if he is accustumed to an opposite one. Cellular prevalence might be defined from a tumour perspective or from a sample perspective [**CELPREV**]:  
   Sample cellular prevalence – fraction of cells that contain the mutation in the entire sample; whereas:  
   Tumour cellular prevalence – fraction of cells that contain the mutation in the cancerous cells only;  
   In PhyloWGS, cellular prevalence is defined from a sample perspective, and our CCF is calculated by dividing the cellular prevalence by the sample purity. The results, demonstrated **in Figure 2** are consistent with the ‘truth’ values supplied by the project supervisor.

It is also important to note that the inclusion for the copy number variations (CNVs) data for running PhyloWGS is optional; skipping it would require one less change to the original source code (one less input parser would be required). However, since it’s a distinctive feature of the tool we chose to include it. We have not made comparisons whether it actually contributes to an improvement in the final results in our case (compared with the ‘truth’ values) as we trusted the findings of the authors of PhyloWGS [**PHYLO**].

Lastly, PhyloWGS does not work well if a majority of MCMC chains report a polyclonal tumour structure and is unable to yeld observable results in these cases **[POLYREMOVE]**. Out of 138 simluated samples we tested PhyloWGS-related part of the pipeline with, PhyloWGS failed to converge in two cases (Samples 117 and 127). We believe this might have been due to:

1. An error in the interim ssm\_data.txt outputs (as reported by other users **[POLYERROR]**)
2. Relatively low parameter values we tested the tool with, due to our hardware limitations.
3. The true nature of the tumours being, in fact, polyclonal.

In case of encountering a polyclonal tumour, the pipeline will report a runtime error during the execution of the native write\_results.py script and the results for those specific samples will not be converted to the common format, but the execution for other samples will continue. Should adjusting the PhyloWGS-related pipeline parameters not solve the issue, we recommend trusting the inference of the other two Phase One tools about those samples.

C:\Users\Lukas Rubikas\Downloads\Untitled Diagram (6) (2).png

Figure 3: The workflow of PhyloWGS-related portion of the pipeline

## References

[**PHYLO**] A. G. Deshwar, S. Vembu, C. K. Yung, G. Jang, L. Stein, Q. Morris, PhyloWGS: Reconstructing subclonal composition and evolution from whole-genome sequencing of tumors, Genome Biology 16 (2015) 35.

[**GSL**]GSL – GNU Scientific Library: <http://www.gnu.org/software/gsl/>

[**GITHUB**] PhyloWGS GitHub repository: <https://github.com/morrislab/phylowgs>

[**BESTTREE**] Q. Morris, On the topic of determening the best PhyloWGS tree: <https://github.com/morrislab/phylowgs/issues/61#issuecomment-308578688>

[**CELPREV**] G. Ha, On the topic of differences between definitions of cellular prevalence: <https://github.com/gavinha/TitanCNA/issues/11#issuecomment-336874279>

[**POLYREMOVE**] Snippets of PhyloWGS source code where the removal of polyclonal trees is evident: <https://github.com/morrislab/phylowgs/search?q=polyclonal&unscoped_q=polyclonal>

[**POLYERROR**] GitHub‘s user suggestion for dealing with PhyloWGS‘s polyclonal trees: <https://github.com/morrislab/phylowgs/issues/95#issuecomment-433264817>

# General

## Conclusion

During Phase One of building the data pipeline, we adapted three separate tools for uncovering the tumour subclonal compositions. Due to our hardware limitations, we were unable to test these tools with the recommended parameter values, however the pipeline is capable of running the tools with any parameter values a user requires. In our testing and evaluation phase, we only tried the tools with relatively low paramater values that our hardware was capable of handling.

Due to the different nature of the algorithms behind each of the Phase One tools and little-to-none adaptability of the same parameter values, running time and accuracy analyses between PyClone, Ccube and PhyloWGS should be taken with a grain of salt. In practise, a fair comparison could only be made using the tools with at least their recommended default parameter values. Other confounding factors might contribute to such analyses, such as:

1. Different programming languages used to implement the tools (Python for PyClone, R for Ccube, Python/C++ for PhyloWGS)
2. Different distribution and area of focus between the tools (PyClone and Ccube are distributed as libraries, PhyloWGS is distributed as an end-to-end piece of software that requires many areas of attention)
3. Diffencies between the levels of support by the tool creators (Some tools might be regularly updated and improved in the future, others might be updated rarely if at all)
4. Differencies between the effciencies in our implementations of the tools.

Regardless of these shortcomings, during our testing of the Phase One tools, we observed that Ccube to be orders of magnitude faster than the PhyloWGS and PyClone, that closely follow each other. Hence, using Ccube, we were able to generate (and bundle with our submission) the cluster cancer cell fractions for the whole of the our dataset (500 tumour samples) to be used in the Phase Two of the project: performing latent Dirichlet allocation and non-negative matrix factorization.

## Future work

For Phase One of the project (running PyClone, Ccube and PhyloWGS) the future work consists of the following:

1. Finishing to vectorize the remaining portions of the code in order to achieve a better scalability.
2. Incorporating design patterns to transform our pipeline into a framework or a template; make adding additional tools or libraries for recovering the subclonal composition of tumours seamless and easy.
3. Finish running all of the 500 simulated samples with all of the three tools.
4. Performing a final, comprehensive code review, changing code style to be more readable for the curious users and consistent with the styling guidelines of Python 2.7 and R.
5. Making the pipeline insensetive to the file naming patterns that were evident to our simulated input data. Right now, the pipeline expects segments data to be named ‘Sample.segments.txt’ and .VCF files to be named ‘Sample.no\_real\_info.vcf’. In reality, only the identifying sample name should matter. We believe that would make our pipeline more flexible for an user to use.
6. For PhyloWGS only: encapsulate the changes that we have made to the original source code of the tool and compile instructions in the Docker file to clone the original repository with the overwrite of the files that we have modified. Note that this could render the related portion of the pipeline unusable should a significant change in the original repository occur.
7. For PhyloWGS only: follow the development of the tool and make significant changes to the pipeline once the authors enable the support of easily readable output files.