Informatika Ingenieritzako Gradua

Degree in Informatics Engineering

Konputazio Zientziak

Computer Science

Gradu Amaierako Lana

Bachelor's Thesis

GeRnika: Simulating, Visualizing and Comparing Tumor Evolution Data

Aitor Sánchez Ferrera

Zuzendaritza
Advisors

Borja Calvo Molinos

Konputazio Zientziak eta Adimen Artifiziala Saila

Department of Computer Science and Artificial Intelligence

Maitena Tellaetxe Abete

Intelligent Systems Group (ISG) - Biodonostia

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Bachelor's Thesis

Degree in Informatics Engineering

Computer Science

GeRnika: Simulating, Visualizing and Comparing Tumor Evolution Data

Aitor Sánchez Ferrera

Advisors
Borja Calvo Molinos
Maitena Tellaetxe Abete

2021

Acknowledgements

Incluso si redactar esta memoria ha sido muy costoso, creo que estas son las líneas que más me va a costar escribir. Durante todo el desarrollo de este proyecto me he sentido rodeado de muchas personas que me han llenado de cariño y apoyo, por lo que quiero dedicarles unas palabras.

En primer lugar, quiero agradecer a mis tutores Borja y Maitena todo el trabajo que han realizado dirigiendo este proyecto. Personalmente, esta aventura me ha hecho crecer mucho a nivel profesional, aunque también me ha hecho sentir muy inseguro en muchas ocasiones. Quiero agradeceros toda la ayuda que me habéis proporcionado y que me hayáis hecho sentir tan arropado. Ha sido un gran placer trabajar con vosotros. Muchas gracias.

Esta facultad me ha dado a personas que han resultado ser muy importantes en mi día a día. A Jose, Ander, Olatz y Josune: gracias por todo vuestro cariño y por alegrarme los días en los que estaba algo desanimado y decaído. Sois personas muy especiales para mí y os guardo un aprecio muy grande. Nos vemos muy pronto.

Tampoco puedo olvidarme de todas las amistades que he hecho durante estos últimos cuatro años. A mis amigos de *Pottokopiak* y de la carrera (especialmente a Diego y Maialen, mis *partners in crime*), la gran familia que me ha dado la representación estudiantil y todas las nuevas amistades que han aparecido en mi vida durante esta etapa. Gracias por haberme hecho sentir tan querido. Ha sido una maravilla poder compartir este tiempo con vosotros.

Por último, quiero dirigirme a tres personas que son muy importantes para mí. Gracias por ser un pilar en mi vida y, sobre todo, por tener el valor de aguantarme todos los días. Aunque no vivamos juntos, nunca nos ha faltado una llamada nocturna para contarnos lo duro y cansado que ha sido el día y poder quejarnos (un poco por vicio) de todo lo que nos pasa. Gracias por haber confiado y creído en mí siempre, incluso cuando ni yo mismo lo he hecho. Gracias a mi madre, mi padre y mi hermano por ser ese sitio al que poder escapar cuando el estrés del día a día me tiene agobiado y simplemente necesito un abrazo. Os quiero.

Abstract

Cancer is a collection of genetic diseases based on the uncontrollable division of cells and their spreading into surrounding tissues, caused by changes in DNA. This process leads to the overcrowding of altered cells that form a mass known as tumor. Nevertheless, all cells in a particular tumor do not have the same characteristics as tumors are formed by diverse clones with different mutations, and therefore different properties and behaviour. As a consequence, this intratumoral heterogeneity entails a problem in regard to the diagnosis and medical treatments that can help to manage the disease, potentially resulting in the failure of therapies and the possible propagation of carcinogenic cells to other organs or tissues, which is known as metastasis. Therefore, the development of methods to study the intratumoral heterogeneity are a hot research topic. This project aims at providing a tool to help researchers to easily simulate tumor data and analyze the results of their approaches for studying the composition and the evolutionary history of tumors.

Contents

List of Figures

List of Tables

List of algorithms

CHAPTER 1

Introduction

Tumors undergo various development processes through which different mutations are accumulated, producing subpopulations of cells with different mutational characteristics called clones [1]. This means that the mutational profiles of the clones of a tumor are different, and therefore their properties, e.g., their growth rate, response to treatment or their ability to metastasize, also differ. As a result, this fact gives rise to the Intra-Tumor Heterogeneity (ITH), which hinders the design of effective medical therapies for treating cancer [2]. Consequently, in order to improve the understanding of the disease, researchers are studying the so-called Clonal Deconvolution and Evolution Problem.

1.1 The Clonal Deconvolution and Evolution Problem

The Clonal Deconvolution and Evolution Problem (CDEP) seeks the identification of the genotypes and the frequencies of the clones present in a series of samples of a tumor, as well as the analysis of the ancestral relations among them. Finding out the composition and the evolutionary history of a tumor eases the design of medical treatments adapted to the particularities of the clones of that particular tumor. Thus, the study of the ITH is not only essential for a better understanding of cancer development, but also certainly helpful for customizing efficient therapies for healing this disorder. However, the evaluation of the algorithms for solving the CDEP presents some complications because, as approaches are usually based on experimental data, we can not know whether their results reflect the real evolution of a particular tumor or not.

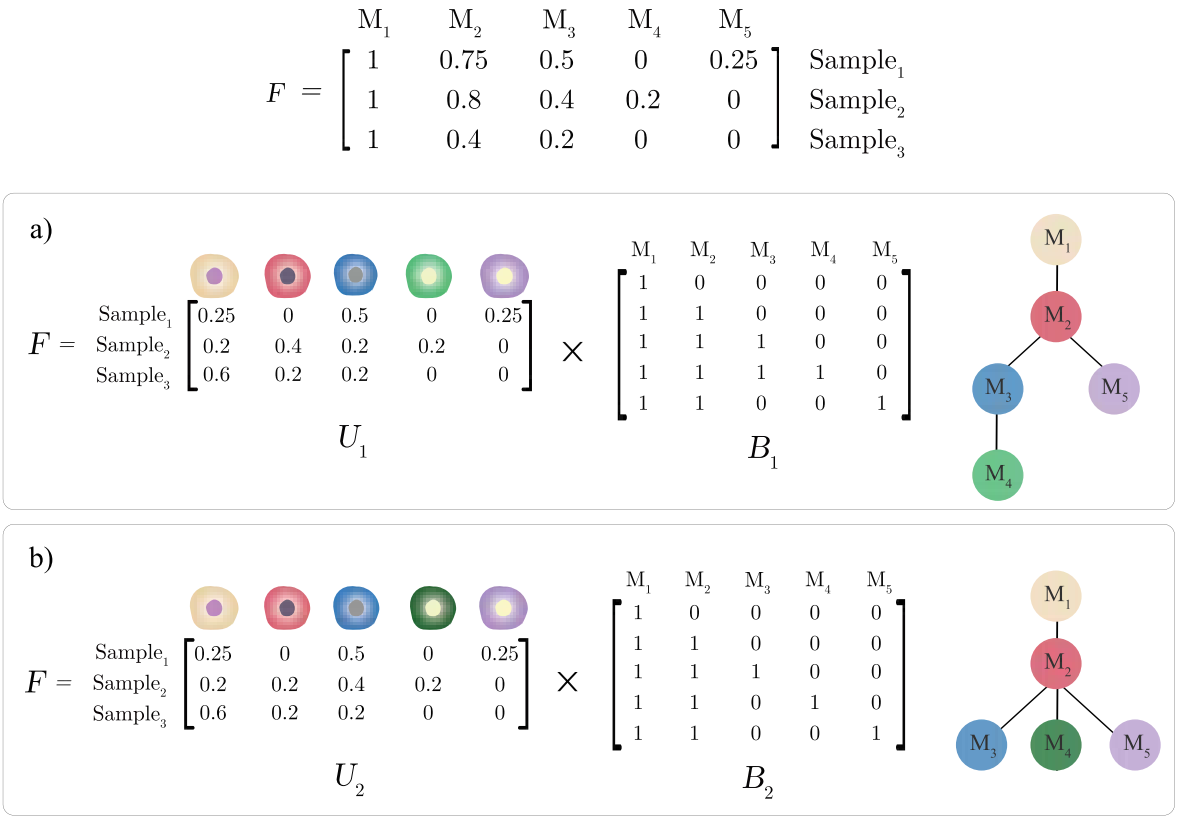
One of the common approaches for solving the CDEP is the resolution of the Variant Allele Frequency Factorization problem (VAFFP), introduced by [3]. According to its formulation, the mutation frequencies observed in a series of tumor samples emerge from the combination of the tumor clonal structure and the clone proportions present in each sample, and it is expressed by way of a matrix decomposition procedure. Broadly speaking, each problem instance consists of a matrix $\boldsymbol{F}$ containing the mutation frequency values in a set of samples –also known as Variant Allele Frequency (VAF) values–, a matrix $\boldsymbol{B}$ that represents the evolutionary relationships among the clones of the tumor and a $\boldsymbol{U}$ matrix that contains the clone proportions in each tumor sample. Subsequently, in accordance with [3], the matrix decomposition is as follows:

$$\boldsymbol{F} = \boldsymbol{U} \cdot \boldsymbol{B} \tag{1.1}$$

Therefore, the VAFFP is based on finding a pair of matrices $\boldsymbol{B}$ and $\boldsymbol{U}$ that produce the observed $\boldsymbol{F}$ matrix containing the VAF values of the mutations present in a series of samples of

a tumor. This problem has been demonstrated to be a NP-complete problem [3]. Figure 1.1 shows two possible solutions for an instance of the VAFFP. In this case, two possible solutions for the $\boldsymbol{F}$ matrix are shown below in panels a) and b). It is visible that the matrices $\boldsymbol{B}$ and $\boldsymbol{U}$ of both solutions present different proportions and evolutionary relationships among clones for the same $\boldsymbol{F}$ matrix.

Figure 1.1 The VAFFP formulation



Overall, solving the CDEP consists of two challenges: resolving the Clonal Deconvolution Problem and reconstructing the phylogeny of the tumor.

1.1.1 The Clonal Deconvolution Problem

The Clonal Deconvolution Problem (CDP) aims at identifying the clones of a tumor and their mutational pattern on the basis of several samples of that particular tumor. This problem outputs the set of mutations found in each sample of the tumor and the estimation of the proportion of cells that contain a particular mutation in each tumor sample, that is, the VAF values of the mutations in that tumor. These values are obtained from the following procedure: first one or more biopsies are sampled from the tumor, then they are sequenced using some bulk DNA sequencing procedure, which profiles a mixture of cells from different clones [1].

The algorithms for solving the CDP can be classified into two groups: (a) exact methods –gathering mathematical optimization and enumeration methods– and (b) probabilistic methods, which compute probability distributions to obtain a solution for this problem[4]. Beyond the approaches, heuristic methods are also a good choice for finding good enough solutions in a more reasonable amount of time than the methods mentioned before.

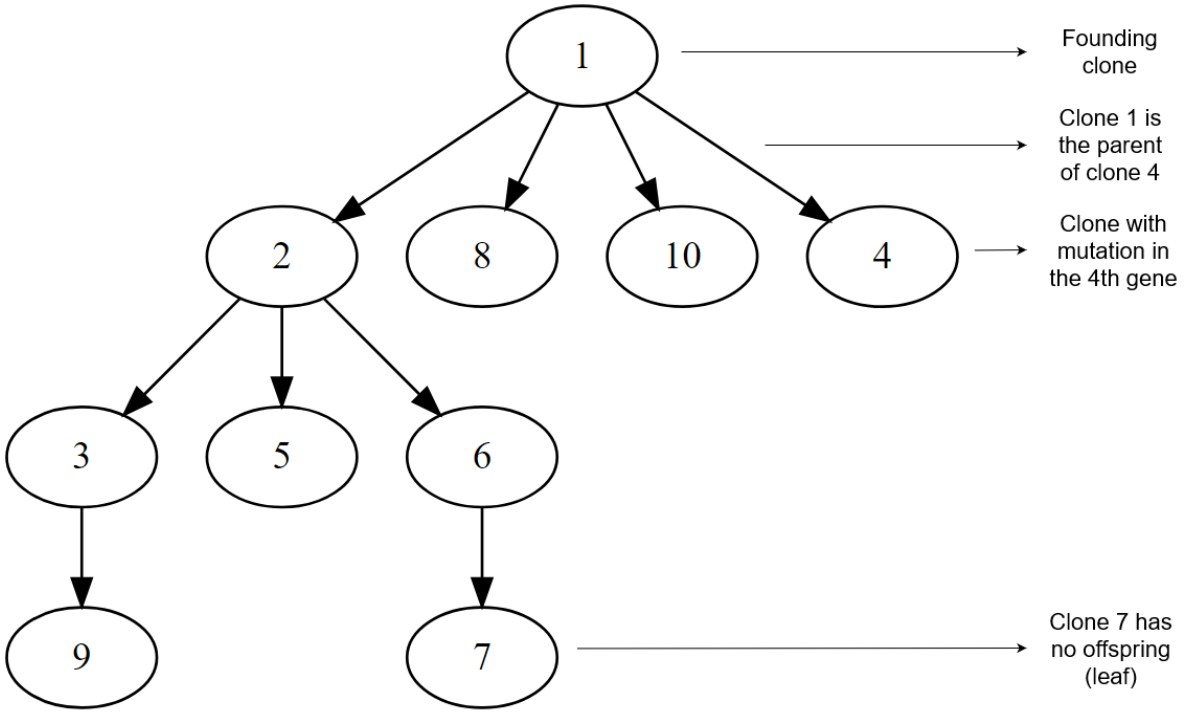
1.1.2 Tumor phylogeny reconstruction

In addition to the study of the composition of tumors, the analysis of the ancestral relationships among their clones is also relevant. In fact, the reconstruction of tumor phylogenies may help identifying the development patterns of that particular tumor in order to predict the evolution of other tumor instances. These relationships may be studied by applying phylogenetic techniques and they can be represented by means of phylogenetic trees.

A phylogenetic tree is a branching diagram that represents the evolutionary relationships among the clones of an organism, in this case a tumor. It is composed of nodes, representing clones that have undergone a mutation at some point in the evolution of the tumor, and edges, which represent the ancestral relationships among the clones [5]. The root of the phylogenetic tree represents the clone where the first mutation of the tumor appeared: the most recent common ancestor of all the tumor clones, also known as the founding clone.

An example of the phylogenetic tree of a tumor composed by 10 different clones may be visualized in Figure 1.2.

Figure 1.2 Phylogenetic tree of 10 nodes



The use of phylogenetic methods is a powerful systems biology approach, as it provides computationally tractable and robust methods for analyzing high-dimensional and heterogeneous cancer data sets [6]. Consequently, several new tools and fresh perspectives are being developed so as to make more realistic prognostics and customize better therapeutic treatments for cancer.

1.2 State-of-the-art tools for studying the ITH and tumor phylogenies

Numerous of the newest and most powerful tools for solving the CDEP and analyzing tumors phylogeny are implemented and gathered in R packages. The reason for this fact is that R is one of the leading programming languages in Bioinformatics, as it is strongly recommended for performing statistics, data analyses and visualisations [7].

Regarding the existing tools for solving the CDP, we may find the following R packages among the ones that use probabilistic methods: `Clomial` [8] (using the Expectation-Maximization algorithm), `sciClone` [9] (based on Bayesian and variational algorithms), `Canopy` [10] (which applies Bayesian algorithms and Markov Chain Monte Carlo methods) and `Cloe` [1] (based on a latent feature model). Note that `Canopy` and `Cloe` are able to estimate tumor phylogenies in order to infer the evolutionary history of tumors.

Additionally, there exists only one R package with wide options for visualizing tumor phylogenies: *ClonEvol* [11]. This package takes the input of a clonal deconvolution tool in order to perform clonal orderings and visualizations in cancer sequencings. However, it does not give the option of comparing different phylogenetic trees nor calculating the distance between them.

All in all, there is a lack of tools that allow users to visualize and compare tumor phylogenies. Moreover, it is necessary to acquire tumor data in order to test and evaluate the performance of the new algorithms that are being developed for studying the ITH, including the CDEP. Consequently, the main aim of this project is to design and implement an R package capable of simulating tumor data, visualizing tumor phylogenies by means of phylogenetic trees, and comparing them by building consensus trees and studying their similarities. The resulting R package of this project is called `GeRnika`.

1.3 Design principles of `GeRnika`

Regarding the principles related to the design of `GeRnika`, it has been implemented in order to be **fundamentally intuitive** and **easy to use**. Furthermore, this package offers **accesible** methods for simulating and analyzing tumor phylogenies by using **simple commands**, **all in one** single package.

In order to implement all its functionalities, `GeRnika` integrates various packages. One of the main principles of `GeRnika` is that, even if it has got some dependencies in regards to the methods of other R packages, it has been designed for **minimizing dependencies** to **mitigate incompatibilities** related to package versions.

Following the above, `GeRnika` incorporates the following packages:

- `data.tree` [12]: General Purpose Hierarchical Data Structure. It is used for building phylogenetic trees through `data.tree` class objects.
- `tidyverse` [13]: Easily Install and Load the 'Tidyverse'. A collection of R packages designed for data science.
- `Diagrammer` [14]: Graph/Network Visualization. It is used for creating and designing consensus trees.
- `MCMCpack` [15]: Markov Chain Monte Carlo (MCMC) Package. It is applied for using statistical models during the simulation of tumor samples.
- `reshape2` [16]: A package for flexibly reshaping data. It is used for reshaping data in order to plot diagrams.
- `colorspace` [17]: A Toolbox for Manipulating and Assessing Colors and Palettes. It is used for customizing default palettes for the methods in `GeRnika`.

CHAPTER 2

Project Management

This chapter presents the planning of the project and the analysis of its evolution, as its management will help us to focus on the work that matters, maintaining under control any deviation that may arise during the project's development.

2.1 Aim of the project

In order to test the algorithms for solving the CDEP and analyzing tumors phylogeny, the goal of this project is to design and implement an R package capable of simulating tumor data, visualizing it by means of phylogenetic tree and comparing different tumor phylogenies.

We expect this tool to be useful for researches related to the solution of the Clonal Deconvolution and Evolution Problem and the study of the ITH, contributing to investigations in the field of oncology. It is remarkable to mention that the resulting R package will be used by the *Intelligent Systems Group* research group –of the University of the Basque Country– for the analysis of a real case of a cancer diagnosis provided by *Biodonostia*.

2.2 Definition of work packages and tasks

In order to achieve the aim of this project, we need to describe the tasks to fulfill during its lifetime. We define the following tasks, classified into the work packages mentioned below.

Management of the project (*WP1*): This involves all the transversal tasks of the project related to its management.

- **Meetings** (*T1.1*): The celebration of periodical meetings with the directors of the thesis to manage the project. Estimated time: 10h.
- **Git repository** (*T1.2*): The creation and the maintenance of a Git repository with all the versions of the project. Estimated time: 5h.
- **Project planning** (*T1.3*): The definition of the tasks and the methodologies used during the project's lifetime. Estimated time: 5h.

Preliminary study (*WP2*): This is focused on gaining a robust theoretical base around some of the core concepts of the project.

- **Biology documentation** (*T2.1*): The general rules of mutations and phylogeny. Estimated time: 10h.
- **CDP documentation** (*T2.2*): The theoretical basis of the CDP and how it is formulated. Estimated time: 10h.
- **R package documentation** (*T2.3*): How an R package is structured and how it may be implemented. Estimated time: 10h.

Implementation of methods (*WP3*): The implementation of the methods of `GeRnika`.

- **Simulation** (*T3.1*): The methods related to the simulation of tumor data were already implemented by the directors of the thesis. This task is based on the adaptation of the previously implemented methods to append them to `GeRnika`. Estimated time: 10h.
- **Visualization** (*T3.1*): Design, implementation and evaluation of methods for visualizing the phylogenetic tree of the sample of a tumor. Estimated time: 40h.
- **Comparison** (*T3.2*): Design, implementation and evaluation of methods for comparing phylogenetic trees and finding their common subtrees. Estimated time: 50h.
- **Consensus** (*T3.3*): Design, implementation and evaluation of methods for creating and rendering consensus phylogenetic trees. Estimated time: 50h.

Implementation and structure of the package (*WP4*): This collects all the tasks related to the implementation of the resulting R package.

- **Structure** (*T4.1*): Designing an efficient structure for the package. Estimated time: 5h.
- **Classification** (*T4.2*): The classification of the methods implemented for the project's final product (i.e., defining the methods useful for users, the ones that are just auxiliary for other methods, etc.). Estimated time: 5h.

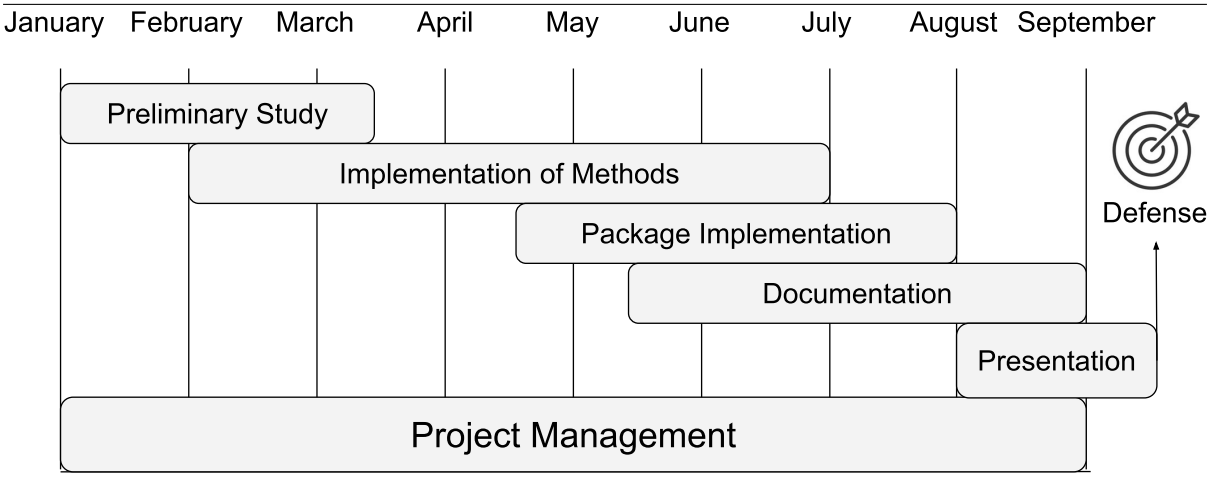
Documentation (*WP5*): The creation of the documentation of the project and `GeRnika`

- **Documentation of the thesis** (*T5.1*): The writing of the document with all the aspects related to the theoretical basis of the project and its development process. Estimated time: 25h.
- **Package documentation** (*T5.2*): The generation of the documentation that will be useful for the future users of the resulting R package (ie.: the *vignettes* of the package, including the experimentation of the methods in the package). Estimated time: 65h.

It is necessary to take into account that the methods related to the simulation of tumor data were already implemented by the directors of the thesis, so we do not need to take into account the task of implementing them in *WP2*.

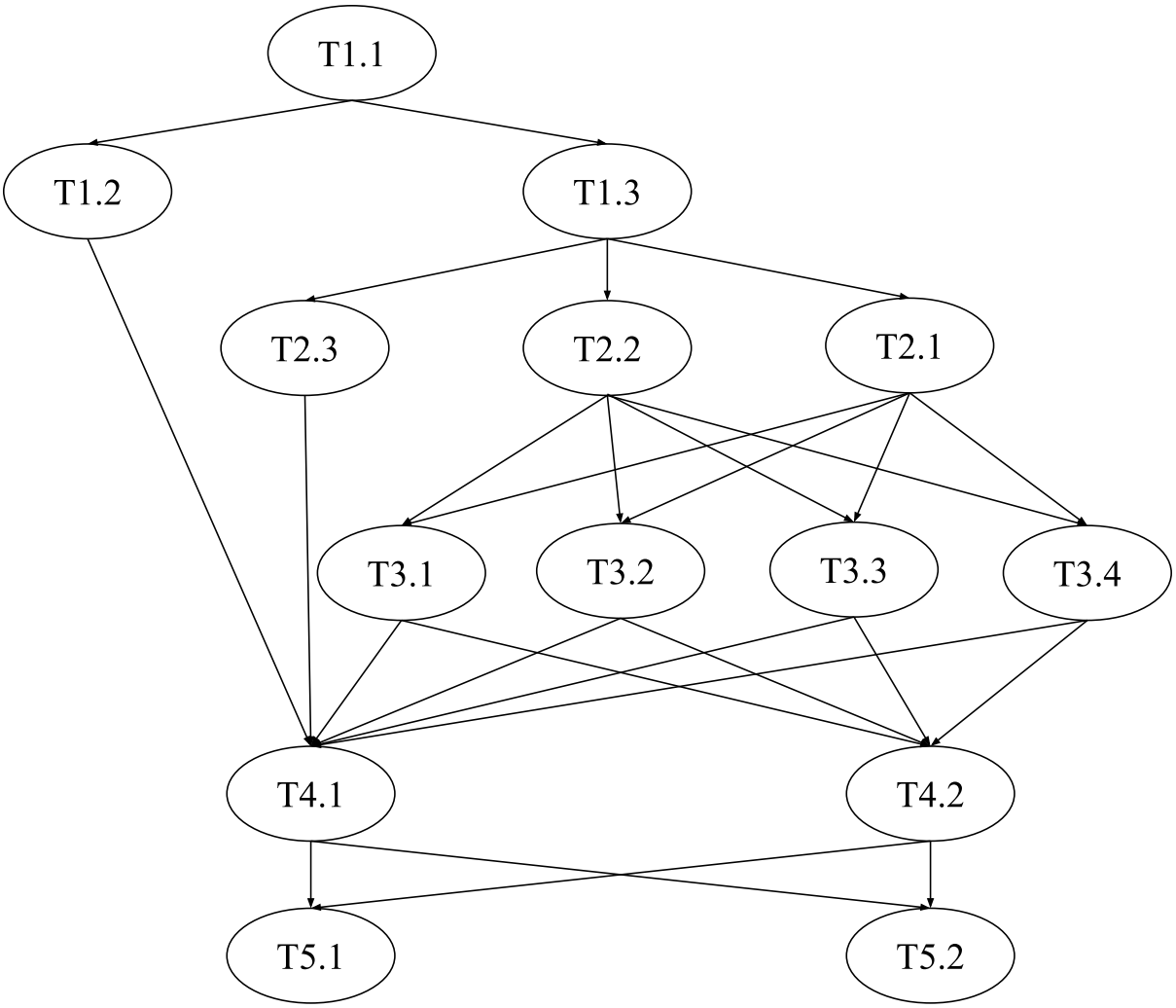
With this in mind, the planning for the work packages described above is depicted in Figure 2.1.

Figure 2.1 Simplified Gantt diagram of the project



Once the work packages and the tasks of the project have been defined, the dependencies among them may be visualized in Figure 2.2, which refers to the task dependency graph of the project.

Figure 2.2 Task dependency graph of the project



2.3 Risk management

The aim of risk management is to identify potential problems that may occur during the project's lifteime. To manage these risks, we will try to study their likelihood of taking place and we will build strategies for mitigating them before they are triggered. Table 2.1 shows the risks that have been identificated, the analysis of their impact and the actions we have taken for mitigating them.

ID	Risk description	Effect	Likelihood	Impact	Mitigating action
1	The theoretical base around the core concepts of the project may not be completely understood	The methods of the package would not be correctly implemented	Medium	High	Asking every single doubt to the advisors of the thesis
2	There may exist incompabilities among the versions of the packages used in the project.	We would not be able to use the methods of some packages	Low	Medium	Searching for other packages that gather similar methods and do not present incompatibilities
3	There may come another lockdown due to the COVID-19 pandemic	We would not be able to access the resources of the faculty for developing the project.	Medium	Medium	Getting installed the software that is needed for the development of the package
4	The author of the thesis or the advisors may be confined	We would not be able to celebrate meetings in person	Medium	Low	Searching for a stable meeting platform apt for at least three people

Table 2.1: Management of the risks of the project

This way, in case any of the previously mentioned risks arise we will be ready to respond to it in order to ensure our project's success.

2.4 Time estimation and deviation analysis

One of the most important assignments of a project's planning is making realistic estimations of the time each of its tasks needs to be fulfilled. Table 2.2 shows the comparison between the estimated time for the tasks of the project and the time they ended up taking.

As the table shows, there are a few differences between the estimated time for some tasks and the time they finally ended up taking. Next, we will explain the reasons that led to these time deviations.

2.4.1 Time deviation for method implementation

As it is represented in Table 2.2, the design and the implementation of the methods for visualizing and comparing phylogenetic trees took less time than it was predicted. The reason for this is that, by using the `data.tree` package it was really intuitive to create the phylogenetic trees of tumor samples.

Nevertheless, it is remarkable that the methods for building consensus trees took more time than we expected. This is because we expent much time looking for a tool capable of creating complex graphs while maintaining their interpretability. At first, we decided to use the *igraph* package, as it offers many simple methods for visualizing complex graphs. However, as the resulting consensus trees were not really intuitive to be interpreted, we decided to use *DiagrammeR*, which took more time as the graphs originated by this tool follow a messy logic and are hard to build.

	Estimation (h)	Real (h)
PROJECT MANAGEMENT		
Meetings	10	10
Git Repository	5	5
Planning	5	5
PRELIMINARY STUDY		
Biology Doc	10	10
CDP Doc	10	10
R package doc	10	10
METHOD IMPLEMENTATION		
Simulation	10	10
Visualization	40	35
Comparison	50	30
Consensus	50	70
PACKAGE IMPLEMENTATION		
Structure	5	5
Classification	5	5
DOCUMENTATION		
Memory of the Thesis	25	25
Documentation of the Package	65	75
Total	**300**	**305**

Table 2.2: "Estimated vs real time" comparison for each task of the project.

2.4.2 Time deviation for documentation

Table 2.2 shows that the creation of the documentation of the package took more time than we firstly anticipated. The reason for this is that, as our aim is to publish the resulting R package of this project, we were compelled to take special care of the resulting *vignettes* and the documentation of the methods gathered in `GeRnika`. The purpose of this fact was to make an easy understanding documentation for the future users of the package.

CHAPTER 3

Statistical and Computational Methods

In the previous chapters we introduced the need of a tool capable of simulating tumor data and analyzing the phylogeny of tumors by means of phylogenetic trees. This chapter explains the algorithms behind the implementation of the different functionalities offered by `GeRnika`.

Firstly, we will explain our approach for simulating tumor data. Then, we will present the logical basis of phylogenetic trees representing the ancestral relations between the clonal subpopulations present. Finally, we will describe a set of techniques for comparing phylogenies.

3.1 Notation

Before going ahead with the explanations of the algorithms in which our methods are based, we will introduce the notation used to describe them.

3.1.1 Numbers, vectors and matrices

Unbolded lowercase x represents a single number, boldface lowercase $\boldsymbol{x}$ represents a vector, and capital boldface $\boldsymbol{X}$ represents a matrix.

Each element in a vector is represented with a subscript that denotes its position in the vector. For instance, x_i refers to the i-th element in vector $\boldsymbol{x}$.

Similarly, an individual element of a matrix is represented with two subscripts denoting the row and column indeces of the element in the matrix. For example, x_{ij} corresponds to the element in the i-th row and j-th column in matrix $\boldsymbol{X}$. On another note, the i-th row entry of a matrix $\boldsymbol{X}$ is represented as $\boldsymbol{x}_{i\cdot}$ and the j-th column of its entry as $\boldsymbol{x}_{\cdot j}$.

3.1.2 Phylogenetic trees and mutations

A clonal tree T that represents the development of a tumor with n mutations is a rooted, directed tree on an n-sized vertex set $V_T = \{v_1, ..., v_n\}$ where v_i corresponds to the i-th vertex or clone. An edge from vertex v_j to vertex v_k represents a direct ancestral relationship (v_j being the parent of v_i) and is denoted as $v_j \rightarrow v_k$. The edges of T are collected on an $n-1$-sized edge set E_T.

The set of n mutations in a tumor is denoted as $M = \{M_1, ..., M_n\}$, where each subscript denotes the mutation index. As there is a one-to-one correspondence between clones and mutations, we can alternatively represent the set of vertices or clones in the tree as either $V_T = \{v_1, ..., v_n\}$ or $M = \{M_1, ..., M_n\}$.

Following, given a clonal tree T, let us define the function $\mathcal{P} : V_T \rightarrow \{V_T, \emptyset\}$, so that $\mathcal{P}(v_i)$ refers to the parent node of v_i. Likewise, let us define another function $\mathcal{K} : V_T \rightarrow \mathbb{P}(V_T)$, so that $\mathcal{K}(v_i)$ refers to the set of children nodes of v_i.

Finally, let Θ_n be the space of phylogenetic trees of size n. Let us then define one last function $\mathcal{R} : \theta_n \rightarrow \{v_1, ...v_n\}$, so that $\mathcal{R}(T \in \Theta_n)$ denotes the root node of T.

3.2 Simulation of tumor data

Essentially, each instance of a tumor simulation consists of a $\boldsymbol{F}$ matrix containing mutation frequency values in a set of samples. In order to build $\boldsymbol{F}$, we will be using a $\boldsymbol{B}$ matrix, representing the phylogeny of the tumor, and a $\boldsymbol{U}$ matrix, which contains the clone proportions in each sample of that particular tumor [4].

This algorithm makes use of three models in order to simulate the previously mentioned matrices: a tumor model simulating the evolutionary history and the current clonal state of the tumor, a sampling model representing the tumor sampling process and a sequencing noise model that adds sequencing noise to the error-free initial VAF values contained in the $\boldsymbol{F}$ matrix.

3.2.1 Tumor model

The tumor model generates a topology T and outputs its associated matrix $\boldsymbol{B}$, jointly with its clone proportions $\boldsymbol{c}$ and the tumor blend at the moment of the sampling.

3.2.1.1 Calculation of the $\boldsymbol{B}$ matrix

Given a number of n mutations, the tumor model iteratively generates a random topology T (which represents the phylogenetic tree of the tumor) with n nodes. To do this, the root node of T is set, with a random mutation as its identifier. Then, each remaining mutation is attached as a new child to an already existing node in T. Additionally, we need to take into account that this model follows two assumptions. On the one hand, we assume that all the clones present in a tumor arise from a single mutation or clone. On the other hand, the output of the model satisfies the Infinite Sites Assumption (ISA), whereby a particular mutation may arise only once in the same tumor, and that mutation can not be lost. Consequently, each new node attached to T inherits all the mutations present in its parent node.

Regarding the attachment of new nodes to the tree, the probability of the nodes in T of being chosen as parent nodes for the new ones is controlled by the topology parameter k. This parameter determines how branchy the topology is. Therefore, as this parameter takes small values, the topology T will be more branched, while the topology will get more linear for bigger values of k. Setting k to 1 generates a completely random topology. Once T is generated, an associated $\boldsymbol{B}$ matrix that represents it is calculated.

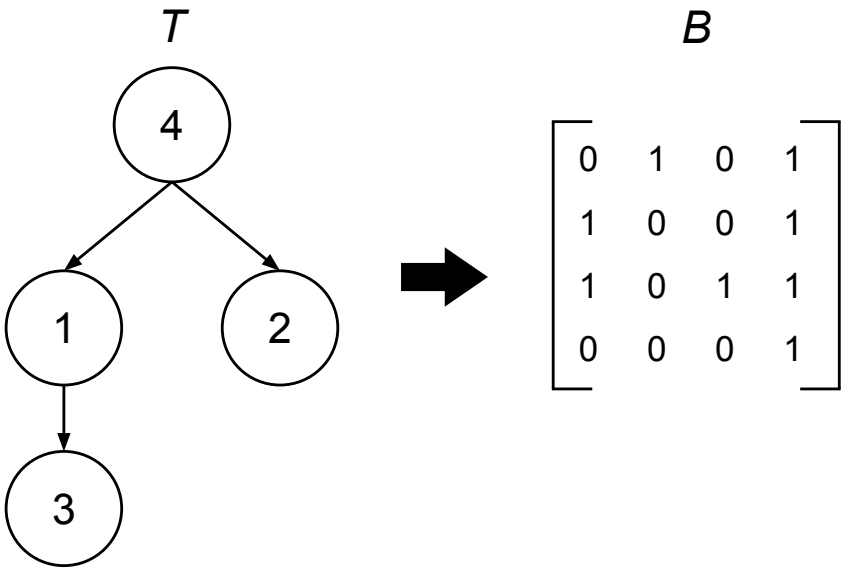
$\boldsymbol{B}$ is a n x n binary clone genotype matrix that represents the mutations present in each clone of a particular tumor. Through this, each row $\boldsymbol{b}_{i.}$ represents the mutations present in clone v_i. According to its interpretation, we conclude that mutation M_j is present in clone v_i when $b_{ij} = 1$. Note that, as our model follows the ISA, the $\boldsymbol{B}$ matrix of a tumor will have a column composed by all ones corresponding to the founding clone. The process for generating $\boldsymbol{B}$ is described in Algorithm 3.1. An example of the calculation of an associated $\boldsymbol{B}$ to T is presented in Figure 3.1.

Algorithm 1 Calculate $\boldsymbol{B}$

Input T
Output $\boldsymbol{B}$
$n := \text{size}(T)$
Initialize identity matrix $\boldsymbol{B}_n$
for i **in** 1:n **do**
  for j **in** the indeces of the descendant nodes of v_i **do**
    $b_{j,i} := 1$
  end
end

Algoritmo 3.1: Algorithm for calculating the $\boldsymbol{B}$ matrix

Figure 3.1 Example of calculating an associated $\boldsymbol{B}$ to T



3.2.1.2 Simulation of the clone proportions $\boldsymbol{c}$

Once T has been generated, the clone proportions of the tumor at the moment of sampling are simulated. To simulate these proportions, the model samples a Dirichlet distribution in each multifurcation of T. For example, for a clone v_i with descendants v_j and v_k, our model will draw a sample(x_i, x_j, x_k) that represents the proportions of those clones from Dir(α_i, α_j, α_k). After that, the proportions of the internal nodes of the tree are scaled to the original proportion of the parent clone, so the sum of the proportions of all the clones in T is one.

In regards to the parameters of the Dirichlet distribution, these depend on the evolution model we assume is followed by the tumor, considering positive selection-driven evolution and neutral evolution. Positive selection-driven evolution involves some mutations having a growth advantage compared to others. Contrarily, neutral evolution entails that no mutations provide fitness advantage and, therefore, different clone subpopulations are present in similar proportions. With this in mind, the $\boldsymbol{\alpha}$ parameters for the Dirichlet distribution have been set to $\boldsymbol{\alpha} = 0.3$ for

the neutral evolution, and $\boldsymbol{\alpha}_{parent} = 5$ for parent nodes and $\boldsymbol{\alpha}_{child} = 10$ for children nodes in the case of positive selection-driven evolution.

3.2.1.3 Simulation of the tumor blend

Finally, the tumor blend, that is how physically mixed are the clones among them, is simulated by modeling the tumor as a Gaussian mixture model of n components. Through this, each component G_i represents a tumor clone and the mixture weights are given by the clone proportions $\boldsymbol{c}$, simulated as described in the previous section.

In this case, the variance for all components is set to 1 and their mean value is set as follows. First, a random clone is selected and the mean value of its component is set to 1. Then, the mean values of the remaining $n-1$ components are calculated sequentially by summing d units to the mean value of the prior component. $d \in (0, 4\}$ is calculated using a Beta distribution as follows:

$$d_{i,j} \sim Beta(1, 5) \tag{3.1}$$

As a result, for $d_{i,j} = 0$ clones i and j are completely mixed, while for $d_{i,j} = 4$ they are physically far from each other [4].

3.2.2 Sampling model

Using the output of the tumor model, the sampling model generates the U matrix of the tumor. Let m and n be the number of tumor samples we want to simulate and the number of clones of that tumor, respectively. Then, $\boldsymbol{U}$ is a m x n matrix that captures the proportion of each clone in each sample, where u_{ij} is the fraction of clone j in sample i. In addition, the values in $\boldsymbol{U}$ must fulfil the sum rule, which is based on the following two conditions. First, as all u_{ij} values represent proportions, they must be non-negative. Secondly, each $\boldsymbol{u}_{i\cdot}$ row denotes the clone proportions in i-th sample, so the values in each row of $\boldsymbol{U}$ must sum up to one [3].

In order to generate $\boldsymbol{U}$, the sampling is performed in a grid-manner over the tumor Gaussian mixture model. Let G_i and G_n be the components with the lowest and largest mean values, respectively. Then, the sampling domain is defined by $[\mu_{Gi} - 3 \cdot \sigma_{Gi}, \mu_{Gn} + 3 \cdot \sigma_{Gn}]$ and divided into $m+1$ equal sized bins representing the sampling sites. Thus, the 1st and m^{th} cutpoints are always set to $\mu_{Gi} - 3 \cdot \sigma_{Gi}$ and $\mu_{Gn} + 3 \cdot \sigma_{Gn}$, respectively, and the remaining $m-2$ are calculated accordingly. Then, for each cutpoint the probability density of the components of the mixture model is obtained and normalized to sum 1. The resulting values $\boldsymbol{p}_i$ are the true tumor clone composition values on that sample. Finally, in order to simulate the physical process of collecting individual cells, the final tumor clone composition values $\boldsymbol{u}_{i\cdot}$ in each sample are modeled by the following multinomial distribution:

$$\boldsymbol{u}_{i\cdot} \sim M(n = 100, \boldsymbol{p} = \boldsymbol{p}_i) \tag{3.2}$$

Note that setting relatively low values for n reduces the number of decimals in the values of $\boldsymbol{U}$. In that manner, clones with really low frequencies may be modeled as absent by setting their composition values equal to 0, which is much more realistic than taking them into account in so low frequencies.

3.2.3 Sequencing noise model

Once we have calculated $\boldsymbol{B}$ and $\boldsymbol{U}$, the $\boldsymbol{F}$ matrix containing the VAF values of the tumor may be computed on the basis of the VAFFP as introduced in the matrix decomposition of Equation

(1.1). As mentioned before, let us have m samples from a given tumor and we sequence them. Now, let n be the number of mutations identified in at least one sample of the tumor. Then, $\boldsymbol{F}$ is a m x n matrix, in which f_{ij} corresponds to the frequency of mutation j in sample i. This third model adds sequencing noise to the f_{ij} values and builds a noisy $\boldsymbol{F}^{\mathrm{n}}$ matrix. Noise is simulated at the level of the sequencing depth, which refers to the average number of reads that map to the same part of the genome μ_{sd}. In order to do this, the sequencing depth r at the genomic position where M_j occurs in sample i is simulated using a negative binomial distribution with parameters $\mu = \mu_{sd}$ and $\alpha = 5$:

$$r_{ij} \sim NB(\mu = \mu_{sd}, \alpha = 5) \tag{3.3}$$

The number of reads that support the alternate allele r_{ij}^{a} is modeled by the following binomial distribution:

$$r_{ij}^{a} \sim B(n = r_{ij}, p = f_{ij}). \tag{3.4}$$

Then, we simulate Illumina mismatch errors, which are known to be around 0.1%. In order to simulate its effect on the VAF values, we estimate the number of reads $r_{ij}^{a'}$ that contain a different allele as a result of a sequencing error as:

$$r_{ij}^{a'} \sim B(n = r_{ij}^{a}, p = 0.001) \tag{3.5}$$

Then, we estimate the number of reads $r_{ij}^{r'}$ of the alternate allele. This parameter represents the reads that contain the reference nucleotide but are read with the alternate allele as a result of this error. Let us imagine that there is one genomic position where the unaltered cells have an A but in some cells there is a mutation and the A changed to a T. For these unaltered cells, with probability 0.1% a sequencing error will occur and instead of a A, one of C, G or T will be read, all with equal chance. Therefore, $\frac{0.001}{3}$% of the times, reads with the mutation of interest (in this case T) will arise from normal reads. The number of reads $r_{ij}^{r'}$ of the alternate allele is estimated as follows:

$$r_{ij}^{r'} \sim B(n = r_{ij} - r_{ij}^{a}, p = \frac{0.001}{3}) \tag{3.6}$$

Finally, the final noisy VAF values f_{ij}^{n} are calculated as:

$$f_{ij}^{n} = \frac{r_{ij}^{a} - r_{ij}^{a'} + r_{ij}^{r'}}{r_{ij}} \tag{3.7}$$

Once this process has been finished, a copy of the initial error-free VAF values are saved in matrix F_true and the resulting error-added VAF values are contained in F. It is remarkable that, even if our simulated data procedure follows the ISA, values in F may break this assumption because of the process of adding noise.

3.3 Visualization of phylogenetic trees

In this section we will explain how to visualize a phylogenetic tree T from $\boldsymbol{B}$. This procedure is based on the interpretation of the B matrix of the tumor representing the relations among the clonal subpopulations that compose it. It must be taken into account that this process does not generate a new phylogenetic tree, as this is already generated and represented in the $\boldsymbol{B}$ matrix after the CDEP has been solved.

First of all, we need to identify the root of the phylogenetic tree $v_r = \mathcal{R}(T \in \Theta_n)$; the founding clone. As this node represents the cell were the first mutation of the tumor arose, we can assume that it contains only that one mutation. Moreover, as we assume that tumors have monoclonal origin, we can conclude that this clone is represented by the only row $\boldsymbol{b}_{r\cdot}$ that contains a single mutation, specifically the mutation present in all the clones corresponding to the column with all ones. The row $\boldsymbol{b}_{r\cdot}$ that represents the root of the phyogenetic tree v_r in $\boldsymbol{B}$ complies with the following condition:

$$\sum_{j=1}^{n} b_{rj} = 1$$

So, if the root of the tree is represented by row $\boldsymbol{b}_{r\cdot}$ and its mutation is contained in clone j ($b_{rj} = 1$), the identifier of the root node representing the first emergent mutation in the tumor will be j.

After setting the root node v_r of the phylogenetic tree of the tumor, we will look for its children nodes $\mathcal{K}(v_r)$. Based on the explanation about the interpretation of the $\boldsymbol{B}$ matrix of a tumor, the child nodes k of the root node v_r must contain the same mutations as its parent plus a new one. In other words, we can infer that $v_r \rightarrow v_k$ for every node v_k that satisfies this condition:

$$\sum_{j=1}^{n} b_{kj} - b_{rj} = 1$$

After finding out which are the children nodes of the root node $\mathcal{K}(v_r)$, we will attach them to the root node and we will repeat this process with the new added nodes. This will be done over and over again until all the clones that compose the tumor are added to the phylogenetic tree, completing T.

The process for obtaining T from $\boldsymbol{B}$ may be checked in Algorithm 2. This process is the inverse of the transformation shown in Figure 3.1.

Algorithm 2 Visualization of a phylogenetic tree from B

Input: $\boldsymbol{B}$
Output: T
root = find_root_node($\boldsymbol{B}$)
T = new_tree(root)
queue = empty_list()
add_element(*queue*, *root*)
Initialize identity matrix $\boldsymbol{B}_n$
while !is_empty(*queue*) **do**
    node = pop(*queue*)
    for *child* in *children* **do**
        attach_node(T, *node*, *child*)
        add_element(*queue*,*child*)
    end
end

Algoritmo 3.2: Algorithm for visualizing a phylogenetic tree from $\boldsymbol{B}$

3.4 Comparison of tumor phylogeny

`GeRnika` offers different functionalities for comparing the phylogeny of tumors. These are based on checking if phylogenetic trees are equal or not, finding the common subtrees between phylogenetic trees and calculating the consensus tree between two phylogenetic trees. This section explains in more detail the methods for comparing tumor phylogenies.

3.4.1 Equal phylogenetic trees

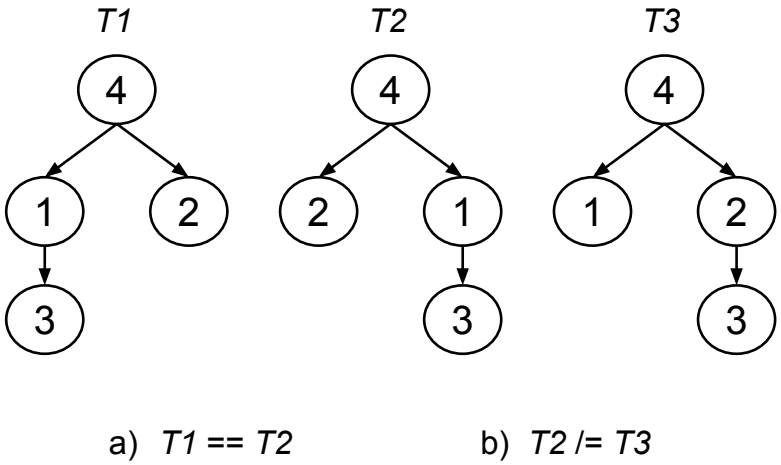
By definition, equal phylogenetic trees are composed by the same nodes, connected by the same edges. For instance, let us have a pair of phylogenetic trees T_1 and T_2 representing two different phylogenetic trees, corresponding to two different tumors that consist of the same set of mutations. Those tumors will share the same ancestral relations between their clones if every node v_i has the same parent node v_j in T_1 and T_2. Thus, the conditions for two phylogenetic trees being equals are represented as follows:

$$V_{T_1} = V_{T_2}$$

$$\forall v_i \in V_{T_1} \Rightarrow \mathcal{P}_1(v_i) = \mathcal{P}_2(v_i)$$

For a better understanding, Figure 3.2 shows two examples of the analysis of the equivalence of various phylogenetic trees. The first example a) shows that $T1$ and $T2$ are equal as they share the same clones related by the same ancestral relationships. Conversely, example b) shows that $T2$ and $T3$ are not equal because they do not share the same evolutionary history (for example, according to $T2$ $\mathcal{P}(v_3) = v_1$ while $T3$ describes that $\mathcal{P}(v_3) = v_2$).

Figure 3.2 Examples of equal and unequal phylogenetic trees



3.4.2 Common subtrees

Nevertheless, the fact of two phylogenetic trees not being equal does not mean that they do not have commonalities, as they may share a set of common subtrees. Let us have the same pair of phylogenetic trees T_1 and T_2 as in the previous example. The intersection $T_1 \cap T_2$ denotes the set

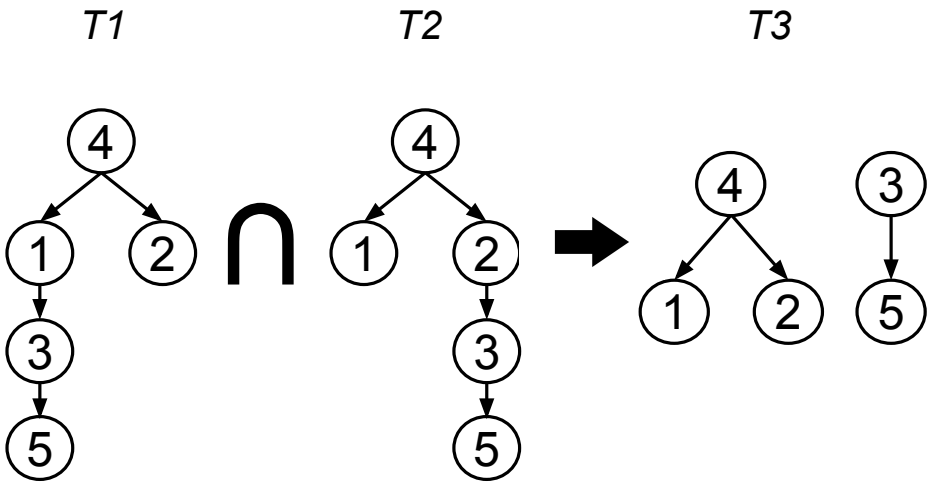
of common subtrees between T_1 and T_2, which are composed by the ancestral relations shared by trees T_1 and T_2. On the one hand, the edges that compose $T_1 \cap T_2$ are the common edges that exist in both trees T_1 and T_2. On the other hand, in regards to the nodes that compose the intersection $T_1 \cap T_2$, these are the ones that are connected by the common edges between T_1 and T_2. Note that the intersection $T_1 \cap T_2$ may be composed by more than one common subtree between trees T_1 and T_2.

This way, the nodes and edges collected by $T_1 \cap T_2$ comply with the condition below:

$$v_i \in V_{T_1 \cap T_2} \iff \exists j (v_i \rightarrow_1 v_j \wedge v_i \rightarrow_2 v_j)$$

Next, Figure 3.3 shows an example of the search of the common subtrees between two phylogenetic trees. In this case, the intersection between $T1$ and $T2$ is composed by a set of two common subtrees.

Figure 3.3 Example of the common subtrees between two phylogenetic trees



3.4.3 Consensus tree

Regarding the search of the common subtrees between two clonal trees, this is not the only way for comparing the commonalities between two different phylogenetic trees that are not equal. It is also possible to combine two phylogenetic trees into a graph that gathers the nodes and the edges of both of them: a consensus tree.

Let us have the same pair of phylogenetic trees T_1 and T_2 as in the previous examples. The union $T_1 \cup T_2$ denotes the consensus tree between $T_1 \cup T_2$, which collects all the ancestral relationships among the clones in both trees T_1 and T_2. The consensus tree between $T_1 \cup T_2$ gathers three type of differentiated relationships between nodes: the common ancestral relationships present in both trees T_1 and T_2 (i.e. the ones that compose their set of common subtrees), the evolutionary relations of T_1 that do not exist in T_2 and the independent ancestral relationships

of T_2. The criteria for classifying these three types of relationships between nodes is described below:

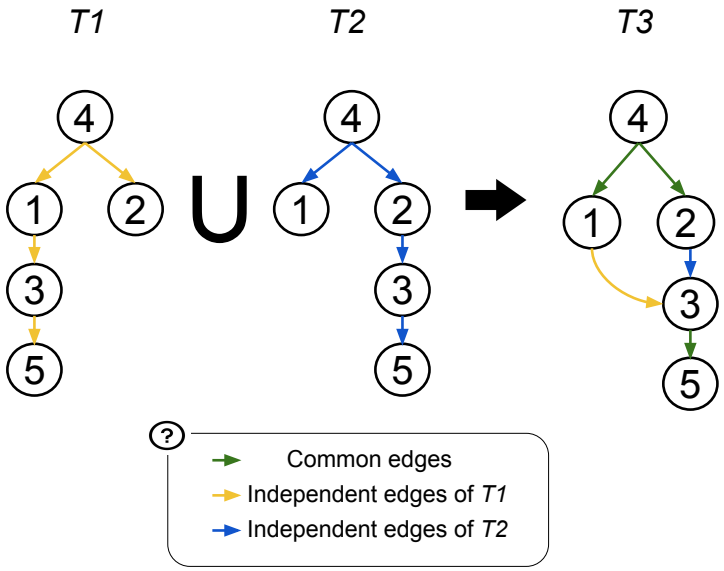
$$\text{Common relationships: } v_i \rightarrow_1 v_j \wedge v_i \rightarrow_2 v_j$$

$$\text{Independent relationships of } T_1\text{: } v_i \rightarrow_1 v_j \wedge v_i \not\rightarrow_2 v_j$$

$$\text{Independent relationships of } T_2\text{: } v_i \not\rightarrow_1 v_j \wedge v_i \rightarrow_2 v_j$$

An example of the visualization of the consensus tree between two phylogenetic trees is presented in 3.4. In this case, we have the same $T1$ and $T2$ trees as in the previous example. It is visible that the common edges that conform the consensus tree $T1 \cup T2$ are the edges that conform the set of common subtrees $T1 \cap T2$. On the other hand, the independent edges of trees $T1$ and $T2$ (i.e. the ones that are not present in $T1 \cap T2$) are represented in different colors.

Figure 3.4 Example of the consensus tree between two phylogenetic trees



3.4.4 Distances

As mentioned previously, manipulating phylogenetic trees based on similarity is essential for many applications. As similarity search has been extensively studied, various similarity measures have been defined, that is, different types of distances between trees. The generally accepted similarity measures for trees in the field of phylogenetics are the Robinson-Foulds (RF) distance [18] and the Tree Edit Distance (TED) [19]. However, it is needed to employ measures apt to compare phylogenetic trees under the perspective of the ITH.

In this case, as we are working with rooted and directed trees, the simplest distance that we may define is based on the quantity of independent edges of the trees that we want to compare. For example, let us have the same pair of trees T_1 and T_2 as in the previous examples. The distance between both trees based on their independent edges d_{diff} may be computed as the sum of the edges of T_1 that do not exist in T_2 and the independent edges of T_2, which is equal to subtracting the number of common edges to the total number of **different** edges (considering

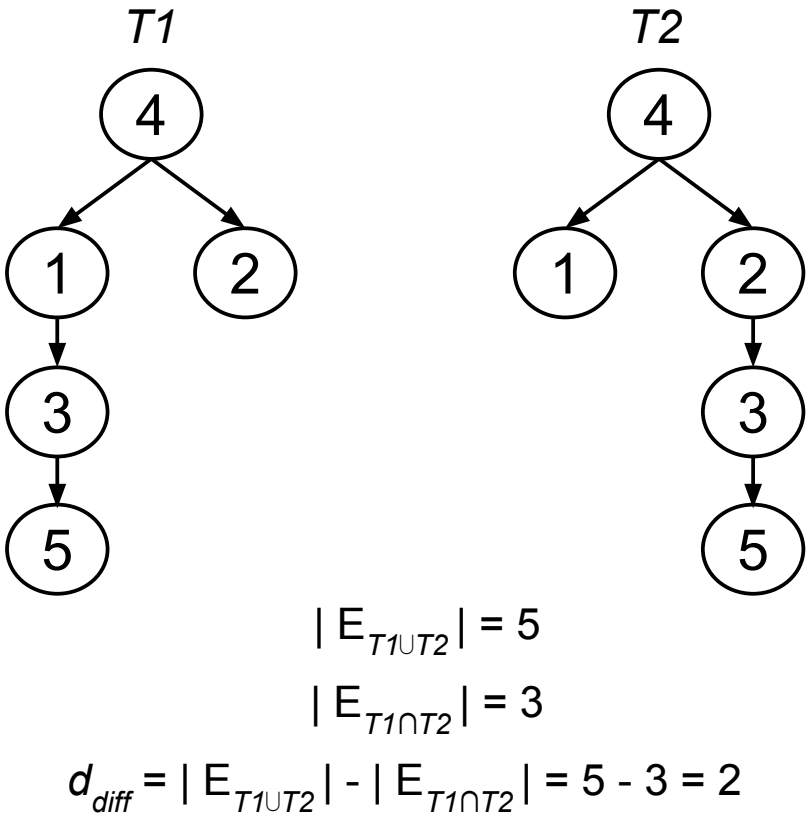
common edges a single one when counting the total number of edges). This distance may be calculated as introduced in Equation (3.8).

$$d_{diff} = |E_{T_1 \cup T_2}| - |E_{T_1 \cap T_2}| \tag{3.8}$$

This way, equal phylogenetic trees with the same ancestral relationships will have $d = 0$ distance according to our measure.

Finally, Figure 3.5 shows an example of the calculation of the distance between two phylogenetic trees.

Figure 3.5 Example of calculating distance between two phylogenetic trees



CHAPTER 4

Package Structure

An R package must follow a clear and intuitive structure that organizes the methods gathered in it according to a predefined logic. Structuring a package also involves defining which are the methods that will be exported for the users' usage and classifying the ones that will be used only as auxiliary functions for other methods [20].

It is remarkable that, as there often exist co-dependencies among the methods in the same package, different structures for a particular package will lead to different hierarchical dependencies among the *source* files in that package. With this in mind, this chapter presents an explanation of the hierarchical strategy that is followed by the distribution of the methods of `GeRnika`. It has to be taken into account that this chapter only presents the structure of the package, as the usage of its methods is explained in Chapter 5.

4.1 The *source code* of `GeRnika`

4.1.1 The distribution

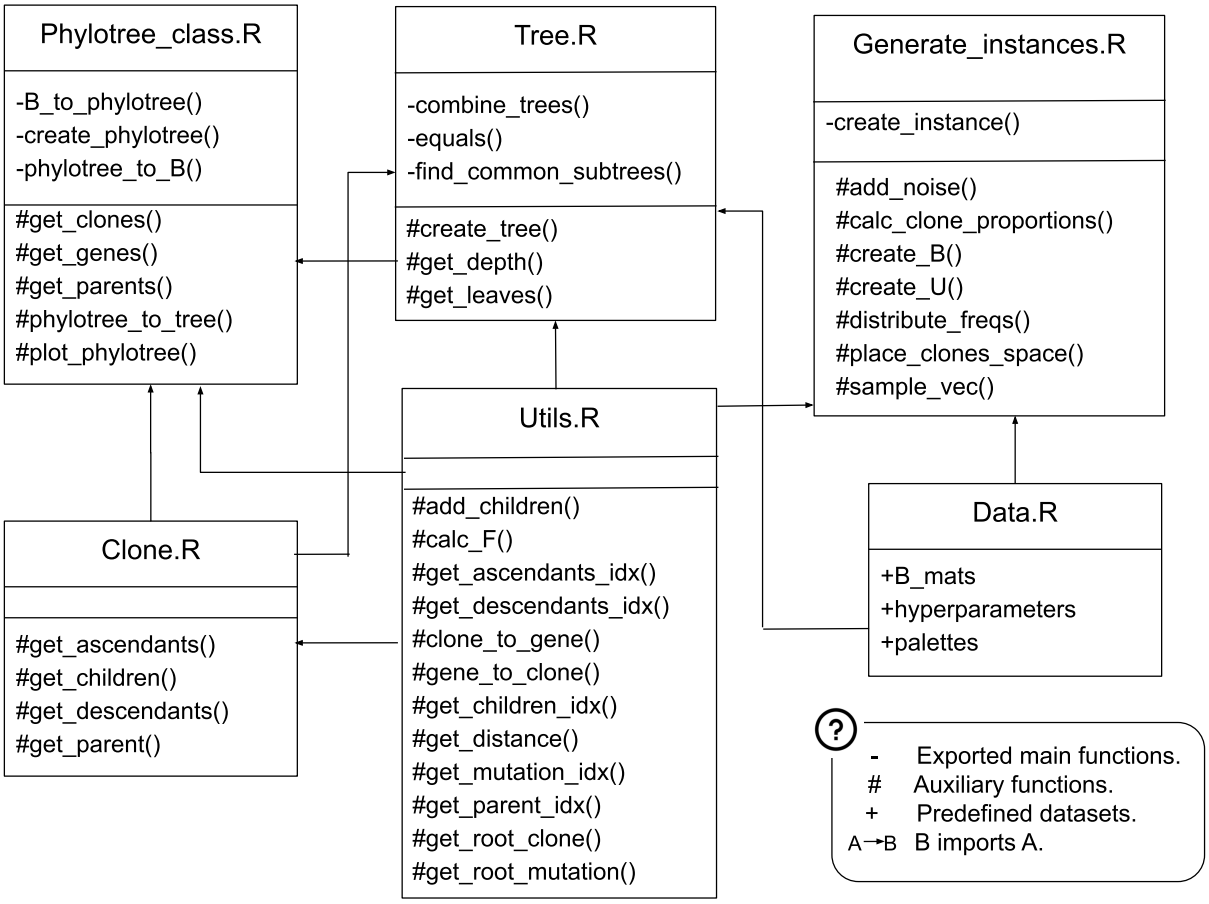
`GeRnika` has been implemented as a *source* package. This kind of packages include particular components, such as a DESCRIPTION file, an R/ directory containing .R files, and a data/ directory that contains a set of predefined datasets for the usage of the functionalities offered by the package. Regarding the distribution of the *source code*, the first principle of making a package is that all R code goes in the R/ directory [20]. In this case, the R/ directory of `GeRnika` consists of six different .R files that collect its methods. These *source files* are outlined below.

- **Generate_instances.R**: This file collects all the methods related to our approach for solving the CDEP and simulating tumor data.
- **Phylotree_class.R**: It contains the methods for defining and instantiating `Phylotree` S4 class objects. The `Phylotree` S4 class is a structure that provides facilities for constructing phylogenetic trees in order to analyze the evolutionary development of tumors. This class is presented in Section 4.1.3.
- **Tree.R**: This file gathers the functions that create the `Node` structures that represent phylogenetic trees. It also includes the methods to compare and combine different phylogenetic trees.
- **Clone.R**: Its methods are auxiliary to the functions contained in *Tree.R* for constructing phylogenetic trees.

- **Utils.R**: This file contains only auxiliary functions for the rest of *source* files of `GeRnika`.
- **Data.R**: It contains the specifications of the predefined datasets of `GeRnika`, which we will introduce later on in this chapter.

As we have mentioned before, the components of `GeRnika` have some dependencies among them, subject to the structure of the package. With this in mind, taking into account the relations among the *source* files of the package, the hierarchy of `GeRnika` is described in Figure 4.1.

Figure 4.1 The dependencies among the *source* files of `GeRnika`.



According to the hierarchy of our package, *Phylotree_class.R*, *Tree.R* and *Generate_instances.R* contain the main methods of the package, that is, the ones that are exported for the users. Contrariwise, *Utils.R* and *Clone.R* gather the auxiliary functions for the methods exported by the main files of the package. Finally, *Data.R* contains the specifications of the datasets exported by `GeRnika`, which we will introduce later on in this chapter.

4.1.2 The exported methods

One of the main tasks when designing an R package is to classify its methods into two groups: the methods that will be exported for the public access, i.e, the ones that enable the functionalities of the package, and the auxiliary functions for the exported methods. The exported functions of a package are the ones that we want other people to use, in other words, the ones that are related to the problem the package is designed to solve. Thus, any functions not related to the purpose of the package should not be exported [20].

This section describes the functions that are exported by `GeRnika` for simulating tumor data and analyzing tumor phylogeny. These functions are summarized below, but their usage is described in Chapter 5.

Create_instance.

- **Description:** This function simulates the instance of a tumor, using the approach described in Section 3.2 for simulating tumor data.
- **Location:** *Generate_instances.R*
- **Input:**

Parameter	Description	Type
n	Number of clones	Discrete No.
m	Number of samples	Discrete No.
k	How branchy the topology is	Continuous No.
selection	The evolution model followed by the tumor	"positive"/"neutral"
noisy	Wheter noise is added to the error-free VAF values or not (optional, TRUE by default) .	TRUE/FALSE
sequencing depth	The average number of reads that map to each locus (for noisy cases, 30.0 by default)	Continuous No.

Table 4.1: Parameters of *create_instance*

- **Output:** The $\boldsymbol{F}$, $\boldsymbol{F\_true}$, $\boldsymbol{B}$ and $\boldsymbol{U}$ matrices that represent the data of a tumor.

B_to_phylotree

- **Description:** This method is the constructor of `Phylotree` class objects on the basis of the B matrix of a tumor sample, based on the approach described in Section 3.3.
- **Location:** *Phylotree_class.R*.
- **Input:** A $\boldsymbol{B}$ matrix representing the ancestral relationships among the clones of a tumor.
- **Output:** A `Phylotree` class object (see Section 4.1.3).

Create_phylotree

- **Description:** This method is the general constructor of `Phylotree` class objects on the basis of its attributes.
- **Location:** *Phylotree_class.R*.
- **Input:** The attributes of the `Phylotree` class, described in Table 4.3.
- **Output:** A `Phylotree` class object.

Phylotree_to_B

- **Description:** This method returns the $\boldsymbol{B}$ matrix of a `Phylotree` class object.
- **Location:** *Phylotree_class.R*.
- **Input:** A `Phylotree` class object.
- **Output:** A $\boldsymbol{B}$ matrix.

Equals

- **Description:** This function checks if two phylogenetic trees are equal or not, based on the approach depicted in Section 3.4.1
- **Location:** *Tree.R*.
- **Input:** Two `Phylotree` class objects.
- **Output:** TRUE or FALSE.

Find_common_subtrees

- **Description:** This function plots the common subtrees between two phylogenetic trees, based on the approach described in Section 3.4.2.
- **Location:** *Tree.R*.
- **Input:** Two `Phylotree` class objects.
- **Output:** A plot and the information about the similarities and the distance between both trees.

Combine_trees

- **Description:** This method builds the consensus graph between two phylogenetic trees, based on the approach described in Section 3.4.3.
- **Location:** *Tree.R*.
- **Input:**

Parameter	Description	Type
Phylotree_1	A phylogenetic tree	`Phylotree`
Phylotree_2	A phylogenetic tree	`Phylotree`
palette	A palette composed by 3 colours	palette

Table 4.2: Parameters of combine_trees

- **Output:** A `dgr_graph` object of the `DiagrammeR` package. We use this class because it gives wide options for visualizing the results of combining two phylogenetic trees into a complex graph, in this case a consensus tree.

It is remarkable that `GeRnika` allows its users to plot phylogenetic trees and consensus trees using predefined tags for tumor clones. This may be done by using the optional argument *labels* = TRUE in functions *find_common_subtrees*, *combine_trees* and the generic *plot* method for the `Phylotree` class.

4.1.3 The `Phylotree` class

`GeRnika` defines a new class, `Phylotree`. The `Phylotree` S4 class is a data structure that provides facilities to represent phylogenetic trees in order to analyze the evolutionary development of tumors. The composition of this class is represented in Table 4.3.

Attribute	Description	Type
B	The square matrix containing the mutations of the clones in the tumor	Matrix
clones	The equivalence table of the clones in the $\boldsymbol{B}$ matrix of the tumor	vector
genes	The equivalence table of the genes in the tumor	vector
parents	The vector of the parents of the clones in the phylogenetic tree	vector
tree	The `Node` structure that represents the phylogenetic tree	`Node`
labels	The tags of the genes in the phylogenetic tree	vector

Table 4.3: The attributes of the `Phylotree` class

Even if `Phylotree` class objects have quite a few attributes, the users of this package will not need to manipulate them as these exist only to reduce the computational cost of the operations between phylogenetic trees. The different ways for instantiating this class are described in Chapter 5.

4.2 Vignettes

Vignettes are a long-form guide to a particular R package [20]. These are like book chapters or academic papers that describe the problems that a particular package is designed to solve, and then show the users how to solve them. In this case, `GeRnika` stores two vignettes: the *paper* vignette (Appendix **??**) and the *usage* (Appendix **??**) vignette. These may be visualized using the *browseVignettes* command for R.

Regarding the *paper* vignette, it introduces the context and the motivation for creating the `GeRnika` package. It provides the instructions for installing the package, its attachment to the user's namespace and it explains the design principles under which the package has been implemented.

On top of that, the *usage* vignette represents a demo for using `GeRnika`. This contains examples to help any user to understand the usage of the functionalities offered by the package in an easy and intuitive way.

Vignettes are a fundamental part of the structure of any R package, and they must be updated regularly as the methods in the package are updated. Consequently, as new functionalities for this package are added, new vignettes that explain how those new functionalities work will be implemented.

4.3 External data

When implementing an R package, it is often useful to include data in it. This data may be used for saving default values for the parameters employed in the different methods of the package [20]. As an alternative, external data may also be included to ease the employment of the functionalities offered by a package.

Depending on the use of the external data, we categorize it into two main groups: exported data and raw data.

4.3.1 Exported data

Exported data is stored in a binary form in the *data/* folder of the package and it is available to the user. `GeRnika` stores the following datasets *exported data*.

- **B_mats**: a `data.frame` containing a set of 10 trios of $\boldsymbol{B}$ matrices. The purpose of saving this dataset in `GeRnika` is to allow its users to use predefined $\boldsymbol{B}$ matrices for trying the methods of the package without the need of creating new ones.
- **hyperparameters**: a `data.frame` containing the static values for the parameters used in the methods of the package.
- **palettes**: a `data.frame` composed by 3 default palettes for the parameters used in the methods of `GeRnika`.

These datasets are documented in the *data.R* file of the package and they may be loaded to the local namespace of the user by using the *GeRnika::«name_of_dataset»* command.

4.3.2 Raw data

As another option, this package also includes non-binary *Raw* data [20]. This data is contained in the *inst/extdata* folder of the package, and it can also be attached by the users of the package by using the *system.file* command. In this case, `GeRnika` stores different files in its *inst/extdata* folder. These include different pictures and animations used for building the vignettes of the package.

CHAPTER 5

Use Case: Analysis of the Effect of the Parameters in the Simulation

This chapter introduces an example of the use of `GeRnika` in R. Following, we present some guidelines to help any user to understand the usage of the functionalities offered by the package, which include the simulation of tumor data, the visualization of phylogenetic trees and the comparison of tumor phylogenies by means of the approaches described in Chapter 3.

Step 1: Simulating tumor data

As explained in Section 3.2, each instance of a tumor's simulation consists of 4 matrices: $\boldsymbol{F\_true}$ (containing the error-free VAF values), $\boldsymbol{F}$, $\boldsymbol{B}$ and $\boldsymbol{U}$. Tumor data is simulated through the *create_instance* function. The information about its parameters and their usage may be checked in Table 4.1. Following, this is an example of the instantiation of a tumor composed by 5 nodes and 4 samples (setting k=0.5 and a "neutral" evolutionary model):

```
I <- create_instance(n=5, m=4, k=0.5, selection="neutral")
```

As a result, this method returns the previously mentioned set of matrices. Once we have shown an example of the instantiation of a tumor, we will analyze the effect of changing the values of the parameters used for its simulation.

The effect of k

k is the parameter that determines whether the topology of a simulated tumor is more or less branched. As a result, higher k values will lead to tumors represented by branchy phylogenetic trees, while lower values of k will produce tumors with more linear phylogenetic trees (more information in Section 3.2.1).

The effect of bigger and smaller values for k on the phylogenetic tree of a simulated tumor is presented below. In order to show the effect of this parameter, we will use `Phylotree` S4 class objects (whose usage will be introduced later in this chapter):

First we will create two new tumor instances $I1$ and $I2$, setting $k_1 = 0$ and $k_2 = 2$:

```
I1 <- create_instance(n=5, m=4, k=0, selection="neutral")
```

```
I2 <- create_instance(n=5, m=4, k=2, selection="neutral")
```

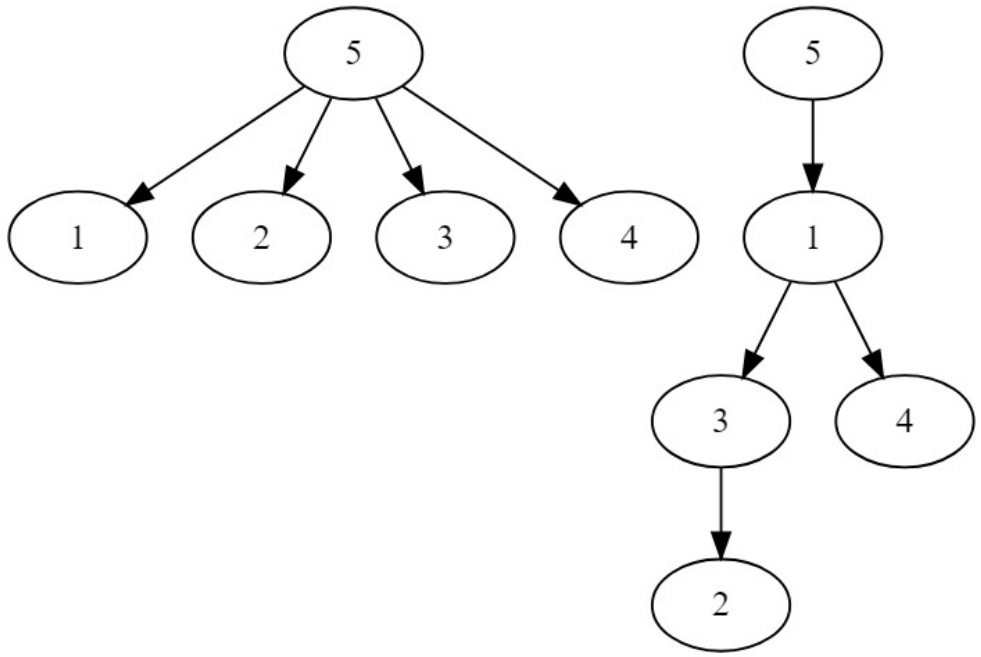
Then, we will instantiate two new `Phylotree` *tree*1 and *tree*2 class objects using the previously simulated tumor instances:

```
tree1 <- B_to_phylotree(B=I1$B)
tree2 <- B_to_phylotree(B=I2$B)
```

Finally, we will visualize the phylogenetic trees of both tumor instances by means of the generic *plot* method for the `Phylotree` S4 class.

```
plot(tree1)
plot(tree2)
```

Figure 5.1 The effect of parameter K.



Following the above, it is visible that the tree on the left ($Tree1$) is totally branchy as it is composed by a root connected to all the leaves of the tree. On the right side we can see a linear tree ($Tree2$), whose structure is conformed by two main branches.

After analyzing the effect of parameter k in the creation of a tumor instance, we will proceed to check the difference between the clonal subpopulations of tumors depending on the evolution model they follow.

The effect of the evolution model

This parameter depends on the evolution model we assume is followed by the tumor, considering positive selection-driven evolution and neutral evolution (introduced in Section 3.2.1.2). These

parameters influences the proportions of the different clone subpopulations in the tumor, contained in its $\boldsymbol{U}$ matrix.

We will create two instances *Ipos* and *Ineu* setting their evolution models to "neutral" and "positive", respectively:

```
Ipos <- create_instance(n=5, m=8, k=0.5, selection="neutral")

Ineu <- create_instance(n=5, m=8, k=0.5, selection="positive")
```

Then, we will show the heatmaps of their $\boldsymbol{U}$ matrices by using the following functions:

```
U_to_heatmap <- function(U, values = TRUE)
  Upos <- melt(U)
  colnames(Upos) <- c("samples", "clones", "proportion")){
  colnames(Upos) <- col_names
  Var1 <- col_names[1]
  Upos<-ggplot(Upos, aes(x=samples, y=clones, fill=proportion)) +
        geom_tile(col="black") +
        scale_fill_gradient(limits=c(0.0000000000001, 1))  +
        theme(axis.title.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.title.y=element_blank(),
        axis.ticks.y=element_blank())
  if(values){
    Upos <- Upos + geom_text(aes(label=proportion), size=4)
  }
  Upos
}

U_to_heatmap(Ipos$U)
U_to_heatmap(Ineu$U)
```

Remembering the explanation of Section 3.2.2, the $\boldsymbol{U}$ matrix of a simulated tumor instance represents the proportion of every clone subpopulation in each particular sample of the tumor. Regarding Figure 5.2, the first heatmap refers to the $\boldsymbol{U}$ matrix of an instance with a neutral evolution and the second one to the $\boldsymbol{U}$ matrix of an instance with a positive selection-driven evolution.

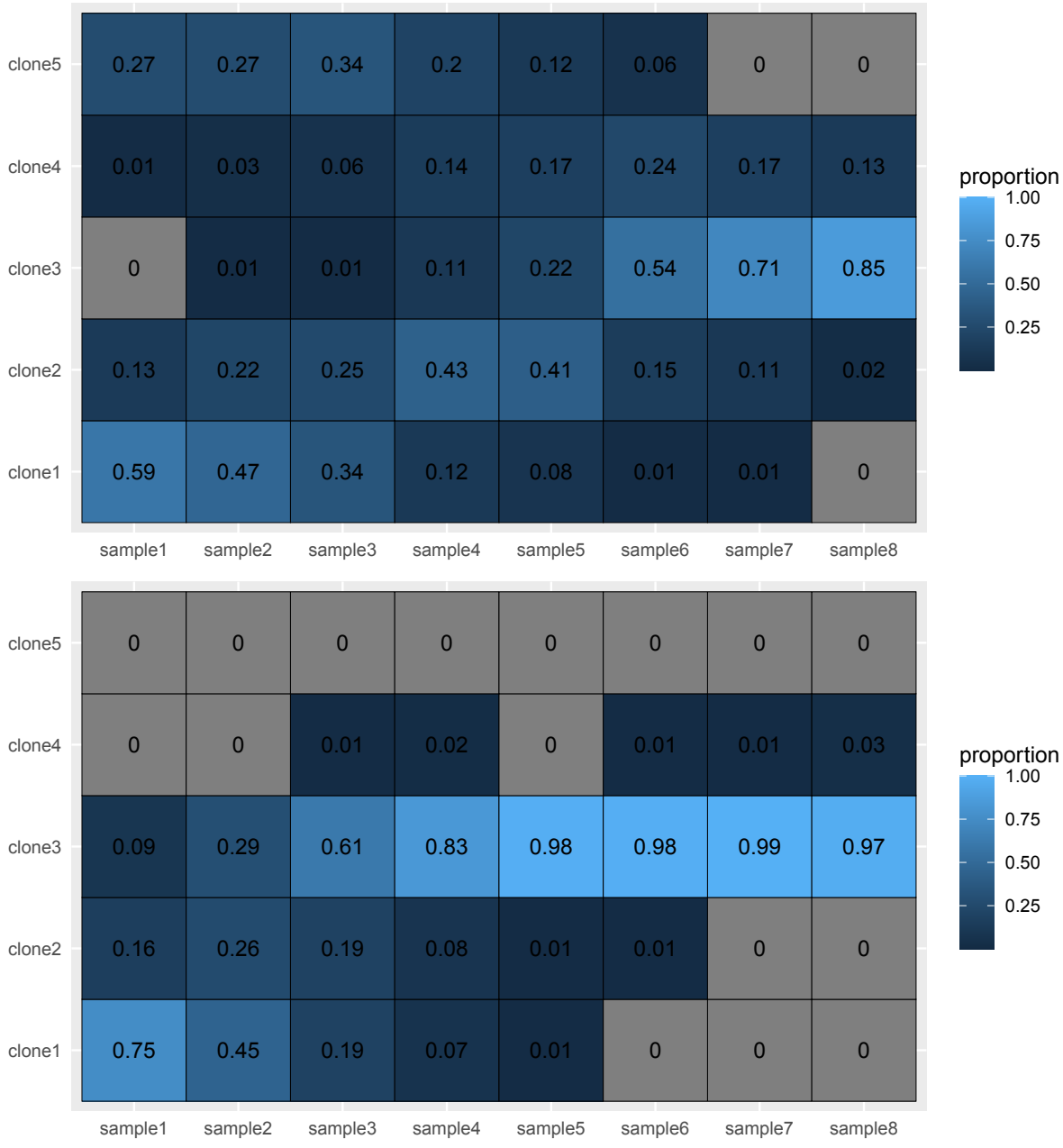
Consequently, we may see that, even if all the different clone subpopulations are not present in similar proportions, almost all clones are present in all the samples of the tumor that follows a neutral evolution model. Conversely, the second heatmap presents that some clones take the biggest part of the tumor samples, as clone 3 provides a growth advantage whereas other clones do not. In addition, this second heatmap shows that there are clones that are missing in more than one tumor sample (For instance, the 5th clone is missing in all the samples of the tumor).

Once we have analyzed the difference between neutral and positive selection-driven evolution models, we will show the results of adding noise to our simulated tumor instances.

The effect of noise

The aim of this process is to add sequencing noise to the error-free Variant Allele Frequency (VAF) values present on the *F_true* matrix of our instance. Then, the resulting VAF values

Figure 5.2 The effect of the evolution model.



will compose the F matrix of our tumor instance.

Now, we will show the difference between error-free and noisy instances by comparing their ***F*** and ***F_true*** matrices. First we will create two tumor instance $Ifree$ and $Inoisy$. $Ifree$ will not have added noise, while $Inoisy$ will have it ($depth = 5.0$):

```
Ifree <- create_instance(n=5, m=8, k=0.5, selection="neutral", noisy=FALSE)

Inoisy <- create_instance(n=5, m=8, k=0.5, selection="positive", noisy=TRUE,
                          depth=5.0)
```

Now we will show the heatmaps that represent the difference between the ***F*** and ***F_true*** matrices of our instances $Ifree$ and $Inoisy$:

```
F_to_heatmap <- function(F, values = TRUE){
  Fpos <- melt(F)
  col_names <- c("samples", "mutations", "VAF")
  colnames(Fpos) <- col_names
  Var1 <- col_names[1]
  Fpos<-ggplot(Fpos, aes(x=samples, y=mutations, fill=VAF)) +
        geom_tile(col="black")+
        scale_fill_gradient(limits=c(0.0000000000001, 1)) +
        theme(axis.title.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.title.y=element_blank(),
        axis.ticks.y=element_blank())
  if(values){
    Fpos <- Fpos + geom_text(aes(label=round(VAF, digits = 2)), size=4)
  }
  Fpos
}

F_to_heatmap(abs(Ifree$F - Ifree$F_true))
F_to_heatmap(abs(Inoisy$F - Inoisy$F_true))
```

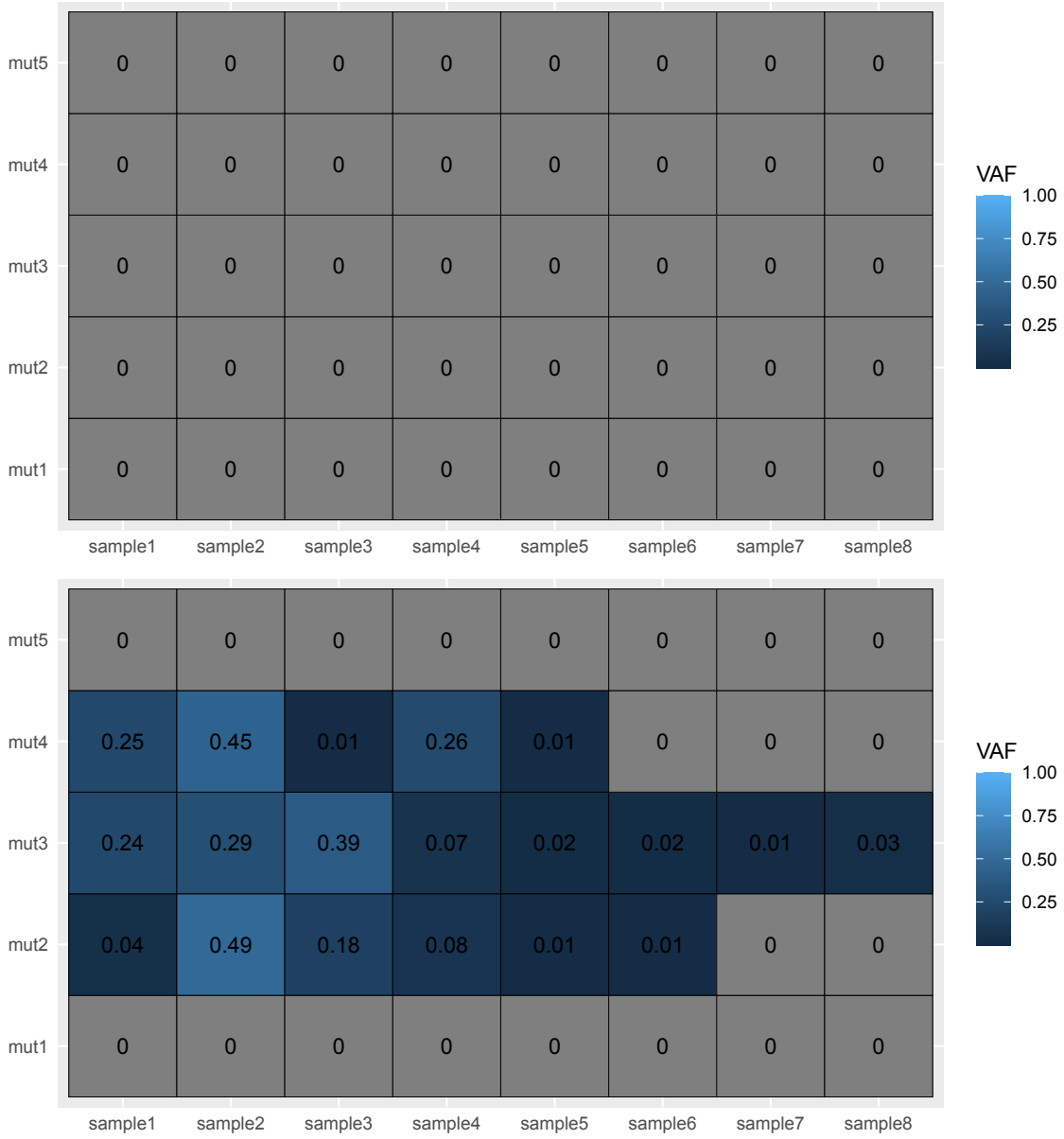
The heatmaps in Figure 5.3 show the **differences** between the ***F*** matrix and the ***F_true*** matrix of each instance, i.e. the noise added to the original VAF values of our tumor samples. The first heatmap presents that there is no difference between the values present in ***F*** and ***F_true***, as it is the one that refers to the error-free instance. Contrarily, the second heatmap shows that the ***F*** and ***F_true*** matrices contain different values, as a result of adding sequencing noise to the initial error-free VAF values of our tumor instance.

It is remarkable that the *create_intance* method allows users to control the sequencing read depth of a simulated instance, which has a direct influence on the resulting VAF values contained in its matrix.

The effect of the depth sequencing

The depth sequencing (or sequencing read) is the average number of reads that map to the same locus (section of the genome). Therefore, higher values for this parameter will produce less noise than lower ones for the original VAF values of a tumor instance. Now we will show the evolution

Figure 5.3 The effect of noise.



of the produced noise-error for instances with a "neutral" and "positive" evolutionary model and depth values between 10 and 5000. We will repeat this process with an instance composed by 10 clones and 2 samples and another one consisting of 100 clones and 10 samples:

```
calc_error <- function(F1, F2) {
  m <- nrow(F1)
  n <- nrow(F2)
  error <- sum(abs(F1 - F2))/(m*n)
  return(error)
}

depths <- seq(10, 5000, 10)
errors_1 <- list()
```

```
errors_2 <- list()
errors_3 <- list()
errors_4 <- list()
for (rep in 1:500) {
      F_10 <- create_instance(n=10, m=2, k=0.5, selection="positive",
              noisy=TRUE, depth=depths[rep])
      F_20 <- create_instance(n=10, m=2, k=0.5, selection="neutral",
              noisy=TRUE, depth=depths[rep])
      F_30 <- create_instance(n=100, m=10, k=0.5, selection="positive",
              noisy=TRUE, depth=depths[rep])
      F_40 <- create_instance(n=100, m=10, k=0.5, selection="neutral",
              noisy=TRUE, depth=depths[rep])
      errors_1[length(errors_1)+1] <- calc_error(F_10[["F"]], F_10[["F_true"]])
      errors_2[length(errors_2)+1] <- calc_error(F_20[["F"]], F_20[["F_true"]])
      errors_3[length(errors_3)+1] <- calc_error(F_30[["F"]], F_30[["F_true"]])
      errors_4[length(errors_4)+1] <- calc_error(F_40[["F"]], F_40[["F_true"]])
}


median_1 <- data.frame(error=errors_1, depth=depths)
median_2 <- data.frame(error=errors_2, depth=depths)
median_3 <- data.frame(error=errors_3, depth=depths)
median_4 <- data.frame(error=errors_4, depth=depths)

color_1 = GeRnika::palette[1]
color_2 = GeRnika::palette[2]

ggplot() + geom_line(data=median_1, aes(x=depth,y=error, color="Positive"),
           size=0.8, show.legend=TRUE) + geom_line(data=median_2,
           aes(x=depth, y=error, color="Neutral"), size= 0.8, show.legend=TRUE)
           + labs(color="Evolution Model")

ggplot() + geom_line(data=median_3, aes(x=depth,y=error, color="Positive"),
           size=0.8, show.legend=TRUE) + geom_line(data=median_4,
           aes(x=depth, y=error, color="Neutral"), size=0.8, show.legend=TRUE)
           + labs(color="Evolution Model")
```

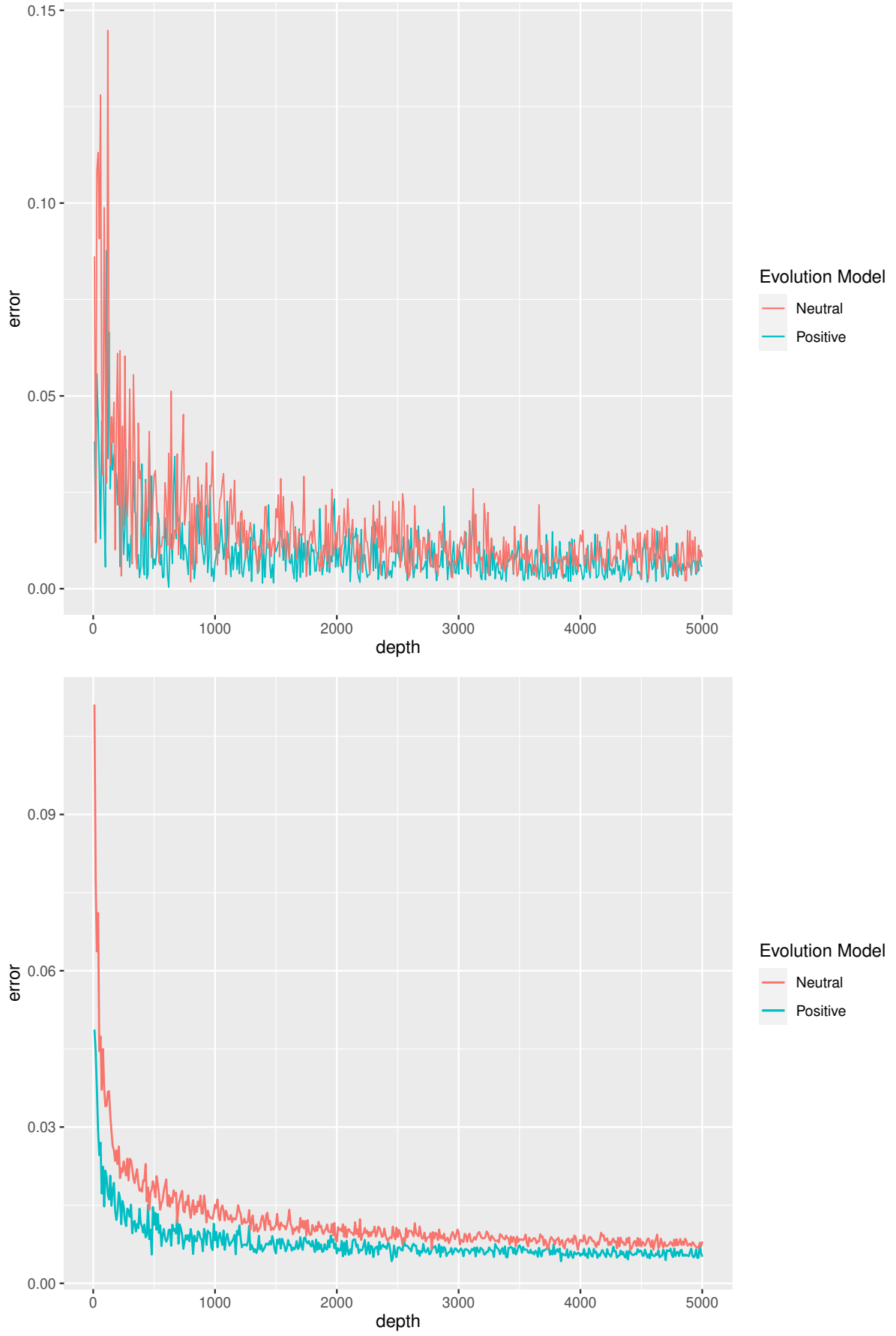
The results of the analysis is described in Figure 5.4. The first graphic presents the progression of the error for a tumor instance composed by 10 mutations and 2 samples. The second graphic shows the same progression for a tumor instance composed by 100 mutations and 10 samples. It is visible that, both instances present a noise-error shrink as depth values get higher, whether they follow a neutral evolutionary selection or a positive-driven one. Note that the VAF values of the second instance are smaller and less changing than the values of the first instance.

After analyzing the effect of the different parameters for creating a tumor instance, we will present the basis of the `Phylotree` S4 class.

Step 2: Visualizing phylogenetic trees

In this section, we will be using the `Phylotree` class for the purpose of inferring and visualizing phylogenetic trees on the basis of simulated tumor data. The `Phylotree` S4 class is a structure

Figure 5.4 The evolution of the error in VAF values based on the read depth sequencing values.



that provides facilities for constructing phylogenetic trees in order to analyze the evolutionary development of tumors. As every S4 class, the `Phylotree` class is composed by various attributes that are essential for building the phylogenetic tree of a particular tumor instance in an optimal way. The attributes of `Phylotree` class may be visualized in Table 4.1.3. The `Phylotree` class is instantiated on the basis of the B matrix of a tumor as presented below.

First, we will create the instance of a tumor with 5 clones, 4 samples, $k = 0.5$ and sequencing noise:

```
instance <- create_instance(n=5, m=4, k=0.5, selection="positive", noisy=TRUE)
```

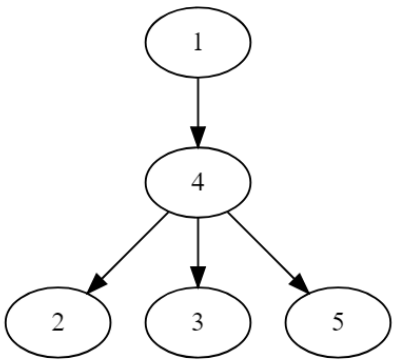
Now we will instantiate a new `Phylotree` class object using the previously generated $\boldsymbol{B}$:

```
phylotree <- B_to_phylotree(B=instance$B)
```

The *B_to_phylotree* method takes a B matrix as an argument and calculates the values for the other attributes present in `Phylotree` class objects. After instantiating the new `Phylotree` class object, we can visualize it using the generic method *plot* for this class:

```
plot(phylotree)
```

Figure 5.5 Phylogenetic tree composed by 5 nodes



Nonetheless, this is not the only way for visualizing the phylogenetic tree of a tumor. Note that this package allows its users to instantiate and visualize `Phylotree` class objects using tags for the names of the clones that compose the phylogenetic tree. This is an example about how this can be done.

we will create a list with the tags we want to insert in our phylogenetic tree and we will use it in order to instantiate a new `Phylotree` class object.

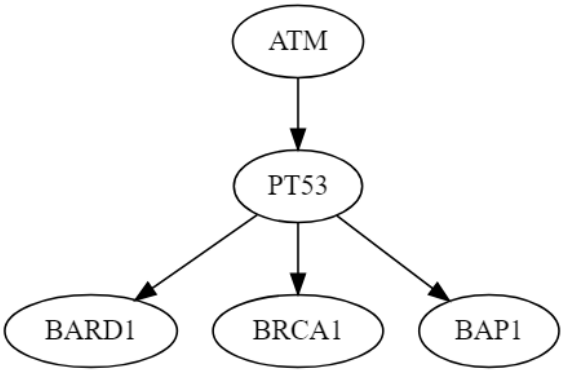
```
tags <- c("ATM", "BARD1", "BRCA1", "TP53", "BAP1")

phylotree <- B_to_phylotree(B=instance$B, labels=tags)
```

After creating the `Phylotree` class object, we may render it using the tags we have previously assigned to the clones in the following way:

```
plot(phylotree, labels=TRUE)
```

Figure 5.6 Phylogenetic tree composed by 5 nodes with tags



This is one of the possible methods for instantiating `Phylotree` class objects on the basis of the B matrix of a tumor. However, this package also grants the option of using the general constructor of the `Phylotree` S4 class for instantiating new `Phylotree` objects, which allows users to give specific values to the attributes of a new `Phylotree` class object.

Once we have shown the usage of the methods for instantiating `Phylotree` class objects and the procedures by these can be visualized, we will proceed to present the functions for comparing the phylogeny of different tumors.

Step 3: Comparing and combining different phylogenetic trees

This package presents different functionalities for comparing the phylogeny of various phylogenetic trees. In order to show how these capabilities work, we will use the *B_mats* dataset of the `GeRnika` package, which contains 10 trios of B matrices based on the solution of various instances of the Clonal Deconvolution and Evolution Problem given by the *ILS* and *GRASP* methods. This trios consist of the following matrices:

- **B_true**: The real $\boldsymbol{B}$ matrix of a simulated tumor instance.
- **B_Grasp**: The initial solution of the *ILS* for finding better solutions for the problem instance. This is generated employing a greedy randomized adaptive heuristic strategy (as introduced by [4]).
- **B_opt**: The optimal solution for the instance of the CDEP, obtained from the *ILS*.

First, we will load the real B matrix of the simulated instance of a tumor and the B matrices given by the *ILS* and the *GRASP* method:

```
B_mats <- GeRnika::B_mats

B_opt <- B_mats[[2]]$B_opt
B_real <- B_mats[[2]]$B_real
```

```
B_grasp <- B_mats[[2]]$B_grasp
```

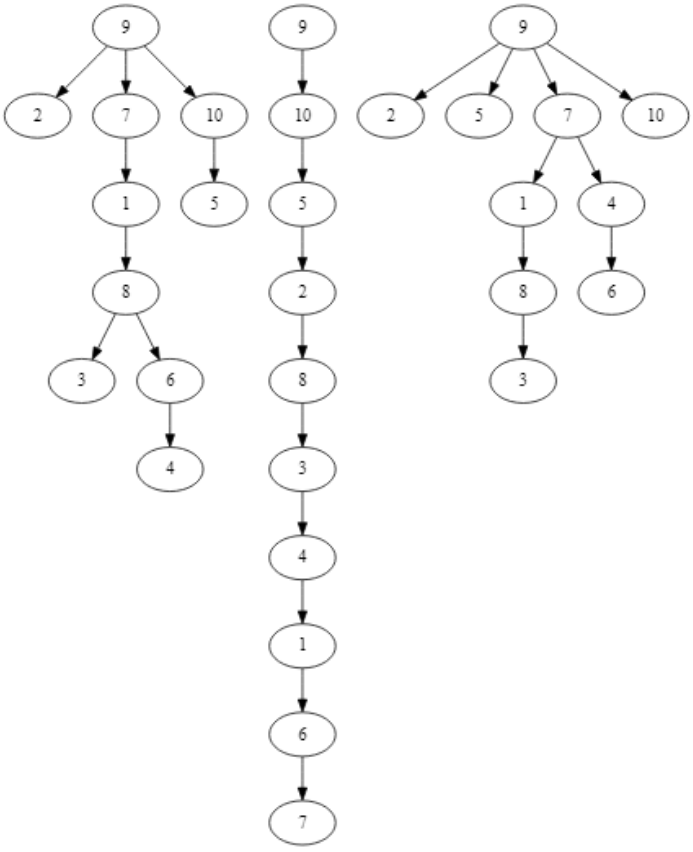
Now, we will use these matrices in order to instantiate new `Phylotree` class objects in order to compare them:

```
tags <- c("ATM", "BARD1", "BRCA1", "TP53", "BRIP1", "CDH1",
            "NF1", "NBN", "PALB2", "PTEN")

phylotree_real <- B_to_phylotree(B=B_real, labels=tags)
phylotree_grasp <- B_to_phylotree(B=B_grasp,labels=tags)
phylotree_opt <- B_to_phylotree(B=B_opt, labels=tags)

plot(phylotree_real)
plot(phylotree_grasp)
plot(phylotree_opt)
```

Figure 5.7 Visualizing *phylotree_real*, *phylotree_grasp* and *phylotree_opt*.



As these three trees above are based on the solution of the same instance for the CDEP, it is reasonable that they are quite similar. Now, we will show the different methods offered by the `GeRnika` package for comparing phylogenetic trees.

The *equals* method

If we compare the phylogenetic trees from above, it is evident that they are not equal. For example, *phylotree_real* and *phylotree_opt* are not equal as some of the edges of *phylotree_real* do not exist in *phylotree_opt* and the other way around.

The equivalence between two phylogenetic trees may be checked by using the *equals* method as follows:

```
equals(phylotree_1=phylotree_real, phylotree_2=phylotree_real)
#> [1] TRUE

equals(phylotree_1=phylotree_real, phylotree_2=phylotree_opt)
#> [1] FALSE
```

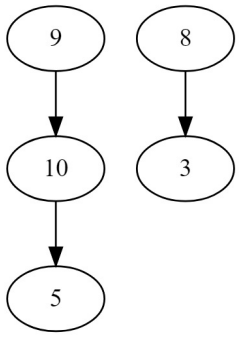
As a result, this method returns *TRUE* when we compare *phylotree_real* with itself. However, as *phylotree_real* and *phylotree_opt* are not equal, this method returns *FALSE* when we check whether they are equal or not.

The *find_common_subtrees* method

In order to find the common subtrees between two phylogenetic trees, the following command may be used:

```
find_common_subtrees(phylotree_1=phylotree_real, phylotree_2=phylotree_grasp)
#> Independent edges of tree1: 6
#> Independent edges of tree2: 6
#> Common edges: 3
#> Distance: 12
```

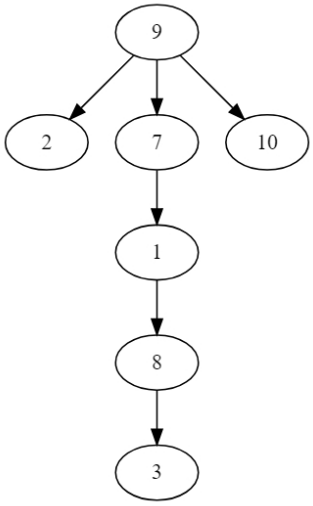
Figure 5.8 The common subtrees between *phylotree_real* and *phylotree_grasp*




```
find_common_subtrees(phylotree_1=phylotree_real, phylotree_2=phylotree_opt)
#> Independent edges of tree1: 3
#> Independent edges of tree2: 3
#> Common edges: 6
#> Distance: 6
```

The *find_common_subtrees* function renders all the common subtrees between two phylogenetic trees. For example, the first call of this method shows that *phylotree_real* and

Figure 5.9 The common subtrees between *phylotree_real* and *phylotree_opt*

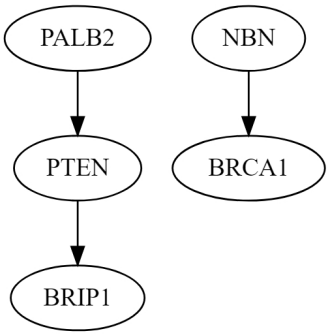


phylotree_opt have a common subtree that covers the biggest part of both phylogenetic trees. In addition, this method prints the information about the similarities and the distance between both trees.

Furthermore, this method provides the option to render the common subtrees between two phylogenetic trees using custom tags for their clones. This can be done in the following way:

```
find_common_subtrees(phylotree_1=phylotree_real, phylotree_2=phylotree_grasp,
                     labels=TRUE)
#> Independent edges of tree1: 6
#> Independent edges of tree2: 6
#> Common edges: 3
#> Distance: 12
```

Figure 5.10 The common subtrees between *phylotree_real* and *phylotree_grasp* with tags

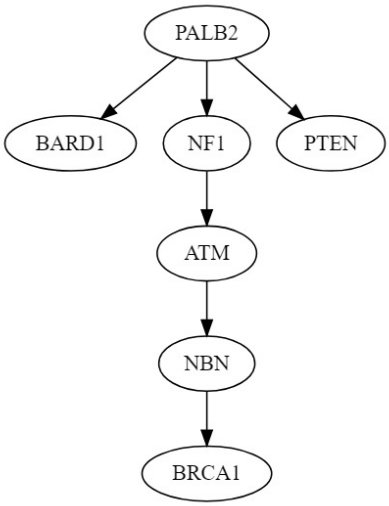



```
find_common_subtrees(phylotree_1=phylotree_real, phylotree_2=phylotree_opt,
                     labels=TRUE)

#> Independent edges of tree1: 3
#> Independent edges of tree2: 3
```

```
#> Common edges: 6
#> Distance: 6
```

Figure 5.11 The common subtrees between *phylotree_real* and *phylotree_opt* with tags



It is perceptible that *phylotree_real* is more similar to *phylotree_opt* than to *phylotree_grasp*. As *phylotree_grasp* represents the initial solution of the Iterated Local Search while *phylotree_opt* describes the optimal solution after performing the ILS, it is reasonable that *phylotree_opt* is more similar to *phylotree_true* than *phylotree_grasp*.

The *combine_trees* method

`GeRnika` package contains a method for combining different phylogenetic trees and building their consensus tree, through which their common edges will be distinguished from the independent edges of each of them.

We will create the consensus tree between *phylotree_real* and *phylotree_opt*. Then, we will build the consensus tree between *phylotree_real* and *phylotree_grasp*. As this method returns `dgr_graph` class objects, we will use the *render_graph* method of the `DiagrammeR` package in order to visualize the consensus trees:

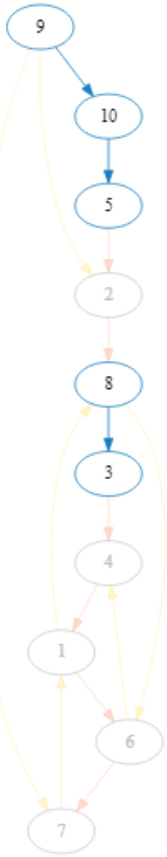
```
consensus_real_grasp <- combine_trees(phylotree_1=phylotree_real,
                                      phylotree_2=phylotree_grasp)

render_graph(consensus_real_grasp)
```

```
consensus_real_opt <- combine_trees(phylotree_1=phylotree_real,
                                    phylotree_2=phylotree_opt)

render_graph(consensus_real_opt)
```

Figure 5.12 The consensus tree between *phylotree_real* and *phylotree_grasp*.



Figures 5.12 and 5.13 present the consensus tree between *phylotree_real* and *phylotree_opt* and the consensus tree between *phylotree_real* and *phylotree_grasp*, respectively. Regarding the trees, the nodes and the edges that compose the common subtrees between the original trees are blue. In addition, yellow edges denote to the independent edges of the tree passed as the first parameter of the method, while orange edges represent the independent edges of the second tree.

Additionally, `GeRnika` gives users the option to build consensus trees using the tags of the clones that compose the phylogenetic trees. Moreover, it is possible to select the palette of colors in which the edges of the consensus tree will be printed. For this purpose, the `GeRnika` package offers three custom palettes to be used for its methods: "Lancet", "NEJM" and "Simpsons". The "Simpsons" palette is used by default for the methods of `GeRnika`.

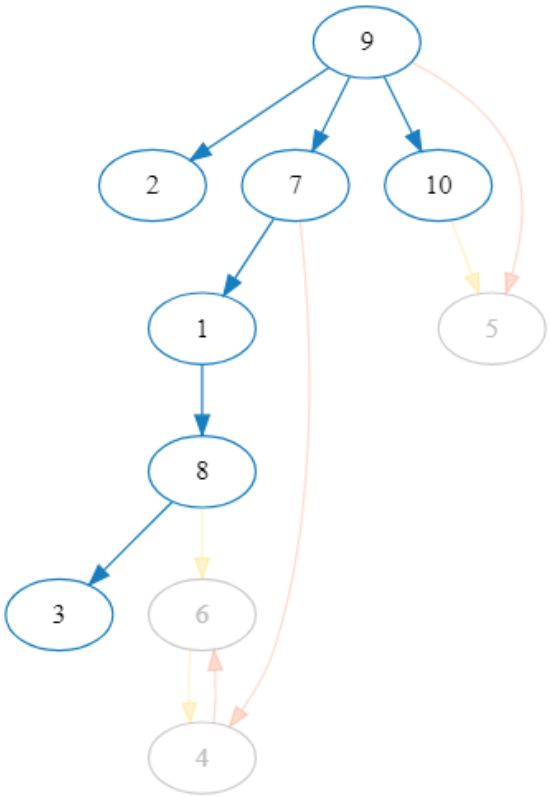
```
palette <- GeRnika::palettes["Lancet"]

consensus <- combine_trees(phylotree_1=phylotree_real,
                           phylotree_2=phylotree_opt,  labels=TRUE,
                           palette=palette)

render_graph(consensus)
```

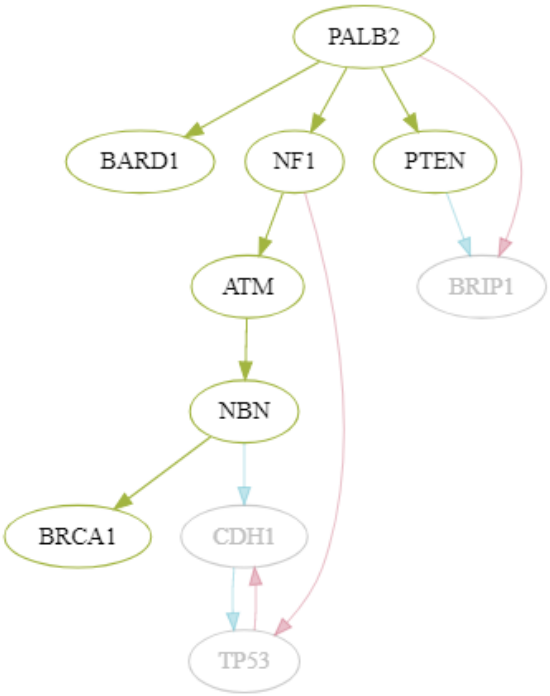
Note that the parameter *palette* of this method may take a palette –a vector containing the hexadecimal code of various colors– composed by three colors in order to use them for building the consensus tree.

Figure 5.13 The consensus tree between *phylotree_real* and *phylotree_opt*.



The consensus tree between *phylotree_real* and *phylotree_opt* using tags and a selected color palette.

Figure 5.14 The consensus tree between *phylotree_real* and *phylotree_opt* using tags and a selected color palette.



Conclusions and Future Work

Once we have explained all the theoretical base behind the implementation of `GeRnika` together with its functionalities and their usage, we will conclude this project by drawing some general conclusions from its results. This chapter also includes the description of the future tasks for completing and improving `GeRnika` and the lesson learned from this project.

General conclusions

We have designed and implemented a tool apt to simulate, visualize and compare tumor data. As new approaches for solving the CDEP are arising, this package will allow researchers to simulate tumor data in order to carry out the experimentation of their algorithms and evaluate their performance. Moreover, `GeRnika` may be really useful for oncologists for analyzing tumor phylogenies in order to customize specific medical treatments for removing tumors before they metastasize. Thus, we can conclude that we have achieved the main aim of this project: to bring in a tool that represents an advancement and a contribution to investigations in the field of oncology.

Furthermore, this package is already available in a public Github repository, so that anybody can download it and employ it for their projects. The instructions for installing it and attaching it to the namespace of `RStudio` are included in the documentation of the package (specifically in Appendix **??**).

Future work

We have realized that there exist many techniques related to phylogenetics in order to analyze the composition of tumors and their phylogenies. This has encouraged us to continue designing and implementing additional methods and functionalities for `GeRnika`. We plan to extend the options for visualizing and comparing different phylogenetic trees.

In regards to the improvement of the visualization of phylogenetic trees, we think that it would be interesting to give users the option to plot phylogenetic trees with different sized nodes depending on their proportions. For instance, clones with higher proportions may be represented with bigger nodes in the tree and clones in lower proportions with smaller ones. On the other hand, as mentioned in 3.8, even if there have been defined various measures for calculating the distance between trees (e.g. the Robinson-Foulds distance and the so called Tree Edit Distance) we have planned to design and implement other measures to compare trees under the perspective of the ITH. It is also remarkable that we plan to publish a paper to promote the use of `GeRnika` and to introduce it to the research community of the field of oncology.

We have also thought about defining different roles for the users of `GeRnika` as we implement new functionalities. For instance, advanced users may be interested in analyzing deeply the ancestral relationships among the clones that compose a phylogenetic tree by accessing to some auxiliary functions (e.g. the *get_descendants* and *get_ascendants* methods).

Finally, in respect of the solutions for the CDEP, we have considered to study another approach for solving this problem by computing probability distributions on trees.

Learning tree distributions: another approach for solving the CDEP

As introduced by [21], Hidden Markov Models (HMMs) are popular models for generating sequential data, which is the simplest form of structured data that consists of a total ordering relation between the atomic elements of the sequence. These models have been generalized in order to learn probability distributions on trees, being trees a form of structured data that represent atomic entities bound by partial order relationships, such as trees in natural language, syntax trees and phylogenetic trees. With this in mind, there exists a class of HMMs that allow learning distributions for tree structured data: Hidden Tree Markov Models (HTMMs). HTMMs model tree distributions by proposing the existence of a hidden generative process by a set of unobserved Markov state random variables.

On the other hand, [22] presents another approach for learning tree structured data: Hidden Tree Markov Networks (HTN). HTNs combine the representation power of generative models for trees and the incremental and discriminative learning capabilities of neural networks by using a neural architecture. As a result, these models allow to learn effective encodings of discriminative structural knowledge using generative tree models while parallelizing their computations.

With this in mind, these models may be really useful in order to infer accurate solutions for an instance of the CDEP on the basis of a set of good solutions for that particular instance of the problem. Hence, we have planned to study this two types of models in order to design and implement another approach for solving the CDEP by computing probability distributions on trees.

Lessons learned

After analyzing the large amount of posibilities for improving and completing `GeRnika`, we have realized that we may enlarge the scope of this project as much as we want, so we can conclude that projects are never finished, but abandoned when we run out of time to continue improving them. Then, the lesson learned from this project is that it is necessary to make the most of the resources we have within reach so as to develop projects that lead to more complete and useful products.

Appendix

Note that the last two appendices of this thesis contain the *paper* and *usage* vignettes of `GeRnika`. However, it must be taken into account that these vignettes are adapted to be exported as *html* format files, so as they are included as *pdf* format files, they have missing various images and animations. The original *html* vignettes are contained in `GeRnika` and they may be rendered by building the vignettes and rendering them through the *build_vignettes* and *browse_vignettes* methods.

Package 'GeRnika'

September 5, 2021

Type Package

Title Simulating, Visualizing and Comparing Tumor Evolution Data.

Version 1.0.0

Author Aitor Sánchez Ferrera, Borja Calvo Molina and Maitena Tellaetxe Abete

Maintainer Aitor Sánchez Ferrera <aitorsanchezferrera@gmail.com>

Description GeRnika is a package capable of simulating tumor data, visualizing it by means of phylogenetic trees and comparing different tumor phylogenies. It aims at providing researchers a tool to easily simulate tumor data and analyze the results of their approaches for studying the composition and the evolutionary history of tumors.

Encoding UTF-8

LazyData true

imports data.tree, DiagrammeR, tidyverse, MCMCpack, reshape2, markdown, knitcitations, colorspace

Depends data.tree, DiagrammeR, tidyverse, MCMCpack, reshape2, markdown, knitcitations, colorspace, R (>= 2.10)

RoxygenNote 7.1.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

License GPL (>= 3)

R topics documented:

B_mats *B_mats is a set of 10 trios of B matrices*

Description

A list of lists composed by 10 trios of B matrices; a real B matrix, a B matrix got by using the Grasp method and another one as a result of an ILS. These matrices can be used as examples for the methods of GeRnika.

Usage

```
B_mats
```

Format

A list of lists composed by 10 trios of B matrices

Trio 1 B_real, B_grasp and B_opt (matrices composed by 5 clones)

Trio 2 B_real, B_grasp and B_opt (matrices composed by 5 clones)

Trio 3 B_real, B_grasp and B_opt (matrices composed by 5 clones)

Trio 4 B_real, B_grasp and B_opt (matrices composed by 5 clones)

Trio 5 B_real, B_grasp and B_opt (matrices composed by 5 clones)

Trio 6 B_real, B_grasp and B_opt (matrices composed by 10 clones)

Trio 7 B_real, B_grasp and B_opt (matrices composed by 10 clones)

Trio 8 B_real, B_grasp and B_opt (matrices composed by 10 clones)

Trio 9 B_real, B_grasp and B_opt (matrices composed by 10 clones)

Trio 10 B_real, B_grasp and B_opt (matrices composed by 10 clones)

Source

Local source; as a result of the Grasp and the ILS methods used for solving the Clonal Deconvolution and Evolution Problem (CDEP).

B_to_phylotree *Create a* Phylotree *object from a* B *matrix.*

Description

Creates a Phylotree class object from a B matrix.

Usage

```
B_to_phylotree(B, labels = NA)
```

Arguments

B	The Matrix that represents the clones in the phylogenetic tree.
labels	Optional argument that refers to the vector containing the tags of the genes of the phylogenetic tree. NA by default.

Value

A Phylotree class object.

Examples

```
# Create a B matrix instance
# composed by 10 subpopulations of
# clones
B <- create_instance(
       n = 10,
       m = 4,
       k = 1,
       selection = "neutral")$B

# Create a new 'Phylotree' object
# on the basis of the B matrix
phylotree <- B_to_phylotree(B = B)

# Generate the tags for the genes of
# the phyogenetic tree
tags <- LETTERS[1:nrow(B)]

# Create a new 'Phylotree' object
# on the basis of the B matrix and
# the list of tags
phylotree_tags <- B_to_phylotree(
                    B = B,
                    labels = tags)
```

---

combine_trees *Get consensus tree between two phylogenetic trees*

Description

Returns a graph representing the consensus tree between two phylogenetic trees.

Usage

```
combine_trees(
  phylotree_1,
  phylotree_2,
  palette = GeRnika::palettes$Simpsons,
  labels = FALSE
)
```

Arguments

phylotree_1	A Phylotree class object.
phylotree_2	A Phylotree class object.
palette	a vector composed by the hexadecimal code of three colors. "The Simpsons" palette used as default.
labels	A boolean, if TRUE the resulting graph will be plotted with the tags of the genes in the phylogenetic trees instead of their mutation index. FALSE by default.

Value

a dgr_graph object representing the consensus graph between phylotree_1 phylotree_2.

Examples

```
# Load the predefined B matrices of the package
B_mats <- GeRnika::B_mats


B_real <- B_mats[[2]]$B_real
B_opt <- B_mats[[2]]$B_opt


# Generate the tags for the genes of
# the phyogenetic tree
tags <- LETTERS[1:nrow(B)]


# Instantiate two \code{Phylotree} class objects on
# the basis of the B matrices
phylotree_real <- B_to_phylotree(
                    B = B_real,
                    labels = tags)

phylotree_opt <- B_to_phylotree(
                    B = B_opt,
                    labels = tags)


# Create the consensus tree between phylotree_real
# and phylotree_opt
consensus <- combine_trees(
               phylotree_1 = phylotree_real,
               phylotree_2 = phylotree_opt)


# Render the consensus tree
render_graph(consensus)


# Load another palette
palette_1 <- GeRnika::palette["Lancet"]


# Create the consensus tree between phylotree_real
# and phylotree_opt using tags and another palette
consensus_tag <- combine_trees(
                   phylotree_1 = phylotree_real,
                   phylotree_2 = phylotree_opt
                   palette = palette_1
                   labels = TRUE)


# Render the consensus tree using tags and the
# selected palette
render(consensus_tag)
```

`create_instance` *Simulate tumor data*

Description

Simulates a tumor instance, composed by `F`, `F_true`, `B` and `U`.

Usage

```
create_instance(
  n,
  m,
  k,
  selection,
  noisy = TRUE,
  depth = 30,
  seed = Sys.time()
)
```

Arguments

`n`	the number of clones.
`m`	the number of samples.
`k`	continuous number how branchy the created topology is
`selection`	character that specifies the clone selection. Possible values: `"positive"` and `"neutral"`
`noisy`	optional logical that specifies whether noise is added to values in `F` or not. FALSE by default.
`depth`	optional argument representing the read sequencing depth (for noisy cases). 30 by default.

Value

the instance of a tumor sample, composed by `F`, `F_true`, `B` and `U` .

Examples

```
# Create an instance composed by 10 clones,
# 4 samples, k = 1, "neutral" selection and
# with added noise and depth = 500
I1 <- create_instance(
  n = 10,
  m = 4,
  k = 1,
  selection = "neutral",
  depth = 500)


# Create an instance composed by 50 clones,
# 10 samples, k = 5, "positive" selection with
# added noise and depth = 500
```

```
I2 <- create_instance(
  n = 50,
  m = 10,
  k = 5,
  selection = "positive",
  noisy = TRUE,
  depth = 500)


# Create an instance composed by 100 clones,
# 25 samples, k = 0, "positive" selection without
# added noise
I3 <- create_instance(
  n = 100,
  m = 25,
  k = 0,
  selection = "positive",
  noisy = FALSE)
```

`create_phylotree` *Create a* `Phylotree` *object.*

Description

The general constructor of the `Phylotree` S4 class.

Usage

```
create_phylotree(B, clones, genes, parents, tree, labels = NA)
```

Arguments

`B`	The Matrix that represents the clones in the phylogenetic tree.
`clones`	numeric vector representing the clones in the phylogenetic tree.
`genes`	numeric vector representing the genes in the phylogenetic tree.
`parents`	numeric vector representing the parents the clones in the phylogenetic tree.
`tree`	`data.tree` object containing the tree structure of the phylogenetic tree.
`labels`	Optional argument that refers to the list containing the tags of the genes of the phylogenetic tree. `NA` by default.

Value

A `Phylotree` class object.

Examples

```
# Create a B matrix instance
# composed by 10 subpopulations of
# clones
B <- create_instance(
       n = 10,
       m = 4,
```

```
        k = 1,
        selection = "neutral")$B


# Create a new 'Phylotree' object
# on the basis of the B matrix
phylotree1 <- B_to_phylotree(B = B)


# Create a new 'Phylotree' object
# with the general constructor of
# the class
phylotree2 <- create_phylotree(
                B = B,
                clones = tree@clones,
                genes = tree@genes,
                parents = tree@parents,
                tree = tree@tree)


# Generate the tags for the genes of
# the phyogenetic tree
tags <- LETTERS[1:nrow(B)]


# Create a new 'Phylotree' object
# with the general constructor of
# the class using tags
phylotree_tags <- create_phylotree(
                    B = B,
                    clones = tree@clones,
                    genes = tree@genes,
                    parents = tree@parents,
                    tree = tree@tree,
                    labels = tags)
```

`equals` *Check if two phylogenetic trees are equal*

Description

Checks wether two phylogenetc trees are equivalent or not.

Usage

```
equals(phylotree_1, phylotree_2)
```

Arguments

`phylotree_1`	A `Phylotree` class object.
`phylotree_2`	A `Phylotree` class object.

Value

A boolean, `TRUE` if they are equal and `FALSE` if not.

Examples

```
# Load the predefined B matrices of the package
B_mats <- GeRnika::B_mats


B_real <- B_mats[[2]]$B_real
B_opt <- B_mats[[2]]$B_opt


# Instantiate two \code{Phylotree} class objects on
# the basis of the B matrices
phylotree_real <- B_to_phylotree(
                    B = B_real)

phylotree_opt <- B_to_phylotree(
                    B = B_opt)


equals(phylotree_real, phylotree_opt)
```

`find_common_subtrees` *find the set of common subtrees between two phylogenetic trees.*

Description

Plots the common subtrees between two phylogenetic trees and prints the information about their similarities and their differences.

Usage

```
find_common_subtrees(phylotree_1, phylotree_2, labels = FALSE)
```

Arguments

`phylotree_1`	A `Phylotree` class object.
`phylotree_2`	A `Phylotree` class object.
`labels`	A boolean, if `TRUE` the rendered graph will be plotted with the tags of the genes in the phylogenetic trees instead of their gene index. `FALSE` by default.

Examples

```
# Load the predefined B matrices of the package
B_mats <- GeRnika::B_mats


B_real <- B_mats[[2]]$B_real
B_opt <- B_mats[[2]]$B_opt


# Generate the tags for the genes of
# the phyogenetic tree
tags <- LETTERS[1:nrow(B)]
```

```
# Instantiate two \code{Phylotree} class objects on
# the basis of the B matrices using tags
phylotree_real <- B_to_phylotree(
                    B = B_real,
                    labels = tags)

phylotree_opt <- B_to_phylotree(
                    B = B_opt,
                    labels = tags)


# find the set of common subtrees between both
# phylogenetic trees
find_common_subtrees(
  phylotree_1 = phylotree_real,
  phylotree_2 = phylotree_opt)


# find the set of common subtrees between both
# phylogenetic trees using tags
find_common_subtrees(
  phylotree_1 = phylotree_real,
  phylotree_2 = phylotree_opt,
  labels = TRUE)
```

`hyperparameters` *hyperparameters for the methods of* `GeRnika`

Description

A data.frame containing the static values for the parameters used in the methods of `GeRnika`.

Usage

```
hyperparameters
```

Format

A data.frame contaning different static values.

Overdispersion value = 0.5

Depth_sequencing value = 30.0

Source

local source; inspired on the optimal parameters for the methods of `GeRnika`.

palettes *palettes for the methods of* GeRnika

Description

A data.frame containing 3 default palettes for the parameters used in the methods of GeRnika.

Usage

```
palettes
```

Format

A data.frame contaning 3 palettes

Lancet #0099B444, #AD002A77, #42B540FF

NEJM #FFDC9177, #7876B188, #EE4C97FF

Simpsons #FED43966, #FD744688, #197EC0FF

Source

Lancet, NEJM and The Simpsons palettes; inspired by the plots in Lancet journals, the plots in the New England Journal of Medicine and the colors used in the TV show The Simpsons, respectively (taken from ggsci package: https://github.com/road2stat/ggsci).

Phylotree_class *Phylotree_class S4 class to represent phylogenetic trees.*

Description

Phylotree_class S4 class to represent phylogenetic trees.

Slots

`B` the data.frame containing the square matrix that represents the clones of the phylogenetic tree.

`clones` a vector representing the equivalence table of the clones in the phylogenetic tree.

`genes` a vector representing the equivalence table of the genes in the phylogenetic tree.

`parents` a vector representing the parents of the clones in the phylogenetic tree.

`tree` a Node class object representing the phylogenetic tree.

`labels` a vector representing the tags of the genes in the phylogenetic tree.

phylotree_to_B *Get B from* Phylotree

Description

Returns the B matrix of a Phylotree object.

Usage

```
phylotree_to_B(phylotree)
```

Arguments

phylotree	a phylotree class object.

Value

A data.frame representing the B matrix of the phylogenetic tree.

Examples

```
# Get the B matrix of a tumor instance
# composed by 10 subpopulations of
# clones
B <- create_instance(10, 4, 1, 1)$B

# Create a new 'Phylotree' object
# on the basis of the B matrix
phylotree <- B_to_phylotree(B)

# Get the B matrix of the phyotree
b1 <- phylotree_to_B(phylotree)
```

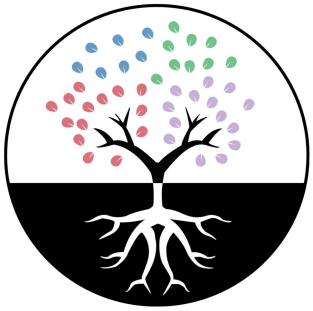
Index

Welcome to GeRnika

Aitor Sánchez, Borja Calvo and Maitena Tellaetxe

2021-09-05

Summary



Cancer is a genetic disease based on the uncontrollable division of cells and their spreading into surrounding tissues, caused by changes in DNA. Nevertheless, all the cells in a particular tumor do not have the same characteristics as this is formed by diverse clones with different mutations, and therefore different properties and behaviour. As a consequence, this intratumoral heterogenity entails a problem in regard to the diagnosis and the medical treatments that can help to manage the disease, resulting in the failure of therapies and the possible propagation of carcinogenic cells to other organs or tissues, which is known as metastasis.

`GeRnika` aims at providing a tool to help researchers to easily simulate tumor data and analyze the results of their approaches for studying the composition and the evolutionary history of tumors. `GeRnika` is expected to be useful for researches related to the solution of the *Clonal Deconvolution and Evolution Problem* (CDEP) and the analysis of tumor phylogenies, contributing to investigations in the field of oncology.

GeRnika package

`GeRnika` may be easily installed with the execution of the following R commands:

```
# commands for installing the package
devtools::install_github("Aitorzan/GeRnika")
```

Once installed, the namespace of `GeRnika` may be loaded by attaching it:

```
library(GeRnika)
#> Loading required package: data.tree
#> Loading required package: DiagrammeR
#> Loading required package: tidyverse
#> -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
#> v ggplot2 3.3.3     v purrr   0.3.4
#> v tibble  3.0.4     v dplyr   1.0.2
```

```
#> v tidyr   1.1.2     v stringr 1.4.0
#> v readr   1.4.0     v forcats 0.5.0
#> -- Conflicts ---------------------------------------- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
#> Loading required package: MCMCpack
#> Warning: package 'MCMCpack' was built under R version 4.0.5
#> Loading required package: coda
#> Warning: package 'coda' was built under R version 4.0.5
#> Loading required package: MASS
#>
#> Attaching package: 'MASS'
#> The following object is masked from 'package:dplyr':
#>
#>     select
#> ##
#> ## Markov Chain Monte Carlo Package (MCMCpack)
#> ## Copyright (C) 2003-2021 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
#> ##
#> ## Support provided by the U.S. National Science Foundation
#> ## (Grants SES-0350646 and SES-0350613)
#> ##
#> Loading required package: reshape2
#> Warning: package 'reshape2' was built under R version 4.0.5
#>
#> Attaching package: 'reshape2'
#> The following object is masked from 'package:tidyr':
#>
#>     smiths
#> Loading required package: markdown
#> Loading required package: knitcitations
#> Warning: package 'knitcitations' was built under R version 4.0.5
#> Loading required package: colorspace
#>
#> Attaching package: 'GeRnika'
#> The following object is masked from 'package:colorspace':
#>
#>     plot
#> The following object is masked from 'package:graphics':
#>
#>     plot
#> The following object is masked from 'package:base':
#>
#>     plot
```

This produces a short report about the versions of the packages that are used by `GeRnika`, providing information about any conflicts with previously loaded packages. `GeRnika` includes the following packages: data.tree (Russ Hyde 2020), tidyverse (Hadley Wickham 2021), Diagrammer (Iannone 2020), MCMCpack (Andrew D. Martin 2021), reshape2 (Wickham 2020) and colorspace (Zeileis et al. 2020)

Design principles

Regarding the principles related to the design of `GeRnika`, this has been implemented in order to be fundamentally **intuitive** and **easy to use**. There exists different *R* packages that allow their users to solve the Cloning Deconvolution Problem (CDP) and analyze the phylogeny of tumor samples, but they do not offer many approaches for visualizing phylogenetic trees in a comprehensive way nor compare the phylogeny of different samples.

Following the above, `GeRnika` offers **accesible** methods for simulating and analyzing tumors phylogeny by using **simple commands**, **all in one** single package.

Acknowledgments

`GeRnika` would not have been possible without the hard work of the *Intelligent Systems Group* members, specially **Borja Calvo** and **Maitena Tellaetxe**, who implemented various methods gathered in this package and coordinated its creation.

References

Abete, Maitena Tellaetxe. 2021. "Metaheuristic Algorithms for the Clonal Deconvolution Problem." *ADDI: Institutional Repository*.

Andrew D. Martin, Jong Hee Park, Kevin M. Quinn. 2021. *MCMCpack: Markov Chain Monte Carlo (Mcmc) Package*. https://cran.r-project.org/package=MCMCpack.

Boettiger, Carl. 2021. *Knitcitations: Citations for 'Knitr' Markdown Files*. https://github.com/cboettig/knitcitations.

Hadley Wickham, Jennifer Bryan. 2015. *R Packages*. O'Reilly Media.

Hadley Wickham, RStudio. 2021. *Tidyverse: Easily Install and Load the 'Tidyverse'*. https://github.com/tidyverse/tidyverse.

Iannone, Richard. 2020. *DiagrammeR: Graph/Network Visualization*. https://github.com/rich-iannone/DiagrammeR.

Russ Hyde, Facundo Munoz, Chris Hammill. 2020. *Data.tree: General Purpose Hierarchical Data Structure*. http://github.com/gluc/data.tree.

Wickham, Hadley. 2020. *Reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package*. https://github.com/hadley/reshape.

Yihui Xie, Garrett Grolemund, J. J. Allaire. 2021. *R Markdown: The Definitive Guide*. Chapman & Hall/CRC.

Zeileis, Achim, Jason C. Fisher, Kurt Hornik, Ross Ihaka, Claire D. McWhite, Paul Murrell, Reto Stauffer, and Claus O. Wilke. 2020. "colorspace: A Toolbox for Manipulating and Assessing Colors and Palettes." *Journal of Statistical Software* 96 (1): 1–49. https://doi.org/10.18637/jss.v096.i01.

Usage of GeRnika

Aitor Sánchez, Borja Calvo and Maitena Tellaetxe

2021-09-05

Introduction

This is a demo for using the `GeRnika` package in R. This document contains examples to help any user to understand the usage of the functionalities offered by the package, which include the simulation of tumor data, the visualization of its phylogenetic tree and the comparison of tumor phylogenies.

Step 1: Simulating tumor data

Each instance of a tumor's simulation consists of a matrix F containing mutation frequency values in a set of samples. In order to build F, we will be using a matrix B, representing the phylogeny of the tumor, and a matrix U, which contains the clone proportions in each particular sample of that tumor.

Tumors can be simulated through the *create_instance* function. The information about its parameters and their usage may be checked in the following table:

Parameter	Description	Type
n	Number of clones	Discrete No.
m	Number of samples	Discrete No.
k	How branchy the topology is	Continuous No.
selection	The evolution model followed by the tumor	"positive","neutral"
noise	Whether noise is added to the error-free VAF values or not (optional, TRUE by default)	TRUE or FALSE
sequencing depth	the average number of reads that map to the same locus (for noisy cases, 30.0 by default)	Continuous No.

Following, this is an example of the instantiation of a tumor composed by 5 nodes and 4 samples (setting k=0.5 and a "neutral" evolutionary model):

```
I <- create_instance(n=5, m=4, k=0.5, selection="neutral")
```

As a result, this method returns the previously mentioned F, B and U matrices and an additional F_true matrix, which we will describe later.

Once we have shown an example of the instantiation of a tumor, we will analyze the effect of changing the values of the parameters used for its simulation.

The effect of k

k is the parameter that determines whether the topology of a simulated tumor is more or less branched. As a result, higher k values will lead to tumors represented by branchy phylogenetic trees, while lower values of k will produce tumors with more linear phylogenetic trees.

The effect of bigger and smaller values for k on the phylogenetic tree of a simulated tumor is presented below. In order to show the effect of this parameter, we will use `Phylotree` S4 class objects (whose usage will be introduced thereupon):

```
# Create an instance of a tumor with k=0:
I1 <- create_instance(n=5, m=4, k=0, selection="neutral")

# Create an instance of a tumor with k=2:
I2 <- create_instance(n=5, m=4, k=2, selection="neutral")

# Create a `Phylotree` class object on the basis of each instance:
tree1 <- B_to_phylotree(B=I1$B)
tree2 <- B_to_phylotree(B=I2$B)

# Plot both trees to check the differences between them:
plot(tree1)
plot(tree2)
```

The effect of parameter K.

Following the above, it is visible that the tree on the left ($Tree1$) is totally branchy as it is composed by a root connected to all the leaves of the tree. On the right side we can see a linear tree ($Tree2$), whose structure is conformed by two main branches.

After analyzing the effect of parameter k in the creation of a tumor instance, we will proceed to check the difference between the clonal subpopulations of tumors depending on the evolution model they follow.

The effect of the evolution type

This parameter depends on the evolution model we assume is followed by the tumor, considering positive selection-driven evolution and neutral evolution. A positive selection-driven evolution model involves some mutations having a growth advantage compared to others. Conversely, neutral evolution models entail that no mutations provide fitness advantage and, therefore, different clone subpopulations are present in similar proportions.

These parameters influences the proportions of the different clone subpopulations in the tumor, contained in its $\boldsymbol{U}$ matrix. The effect of this parameter is described below:

```
# Create an instance of a tumor with neutral selection:
Ipos <- create_instance(n=5, m=8, k=0.5, selection="neutral")

# Create an instance of a tumor with positive selection:
Ineu <- create_instance(n=5, m=8, k=0.5, selection="positive")

# Show the heatmaps of the U matrixes of our instances:
U_to_heatmap(Ipos$U)
U_to_heatmap(Ineu$U)
```

The U matrix of a simulated tumor instance represents the proportion of every clone subpopulation in each particular sample of the simulated tumor. The heatmaps from above are inspired on the U matrices of two different generated instances. The first heatmap refers to the U matrix of an instance with a neutral evolution and the second one to the U matrix of an instance with a positive selection-driven evolution.

Consequently, we may see that, even if all the different clone subpopulations are not present in similar proportions, almost all clones are present in all the samples of the tumor that follows a neutral evolution model. Conversely, the second heatmap presents that some clones take the biggest part of the tumor samples, as clone 3 provides a growth advantage whereas other clones do not. In addition, this second heatmap shows

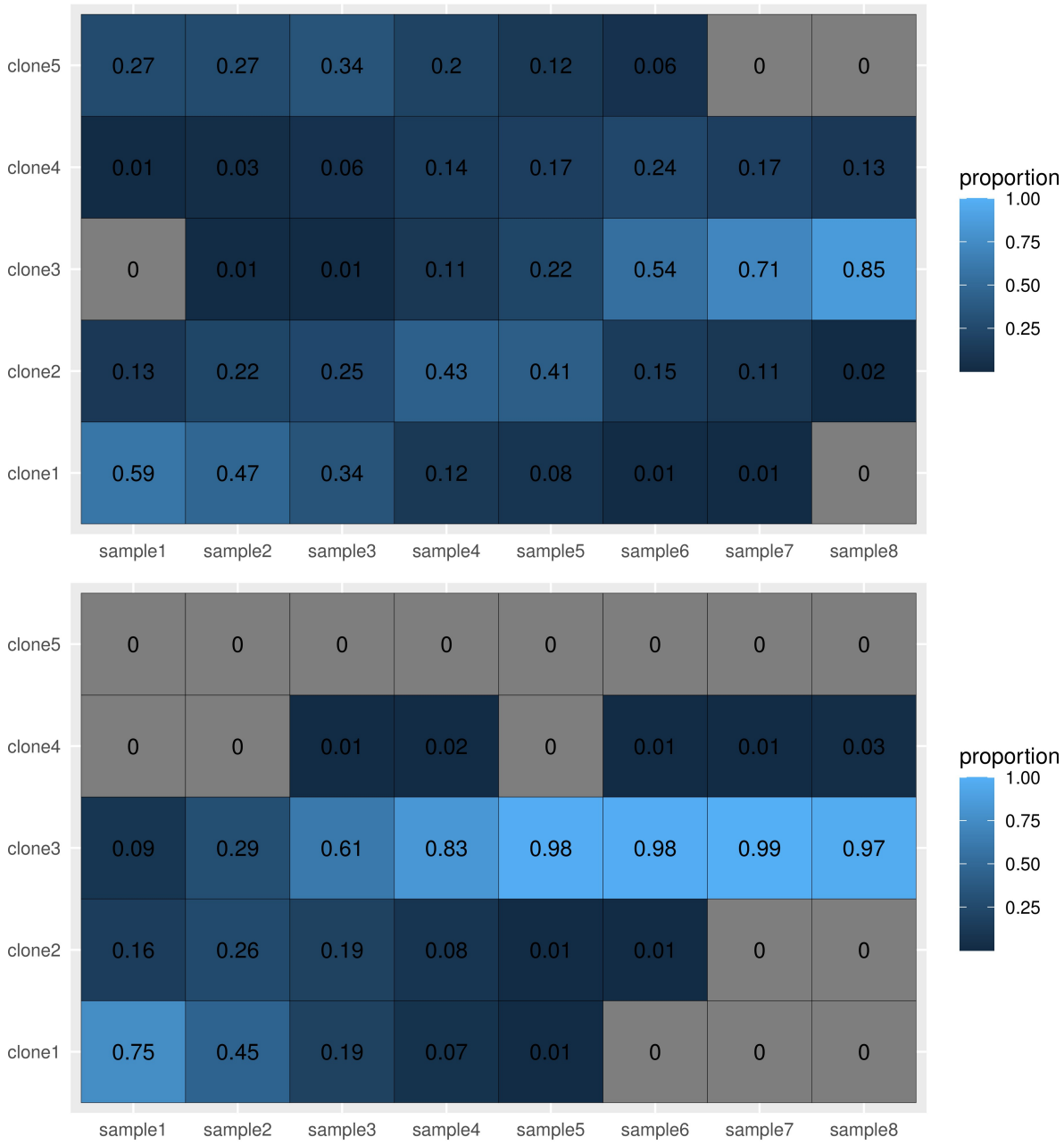


Figure 1: The effect of the selection type.

that there are clones that are missing in more than one tumor sample (For instance, the 5th clone is missing in all the samples of the tumor).

Once we have analyzed the difference between the neutral and positive selection-driven evolution models, we will show the results of adding noise to our simulated tumor instances.

The effect of noise

The aim of this process is to add sequencing noise to the error-free Variant Allele Frequency (VAF) values of our tumor instance, which are contained in the F matrix. The original error-free VAF values are saved in the F_true matrix of the tumor instance.

Now, we will show the difference between error-free and noisy instances by comparing their F_true and F matrices:

```
# Create an instance of a tumor without added noise:
Ifree <- create_instance(n=5, m=8, k=0.5, selection="neutral", noisy=FALSE)

# Create an instance of a tumor with sequencing noise added:
Inoisy <- create_instance(n=5, m=8, k=0.5, selection="positive", noisy=TRUE, depth=5.0)

# Show the heatmaps of the difference between the F and F_true matrixes
# of our instances:
F_to_heatmap(abs(Ifree$F - Ifree$F_true))
F_to_heatmap(abs(Inoisy$F - Inoisy$F_true))
```

The heatmaps from above show the differences between the F matrix and the F_true matrix of each instance, i.e. the noise added to the original VAF values of our tumor samples. The first heatmap presents that there is no difference between the values present in F and F_true, as it is the one that refers to the error-free instance. Contrarily, the second heatmap shows that the F and F_true matrices contain different values, as a result of adding sequencing noise to the initial VAF values of our tumor instance.

It is remarkable that the *create_intance* method allows users to control the sequencing read depth of a simulated instance, which has a direct influence on the resulting VAF values contained in its F matrix.

The effect of the sequencing read depth

The sequencing read depth is the average number of reads that map to the same locus (section of the genome). Therefore, higher values for this parameter will produce less noise than lower ones for the original VAF values of a tumor instance.

See the evolution of the produced noise-error for instances with different depth values below. The first animation below present the progression of the error for an instance composed by 10 and 2 samples, while the second one shows the evolution of the error for an instance composed by 100 clones and 10 samples:

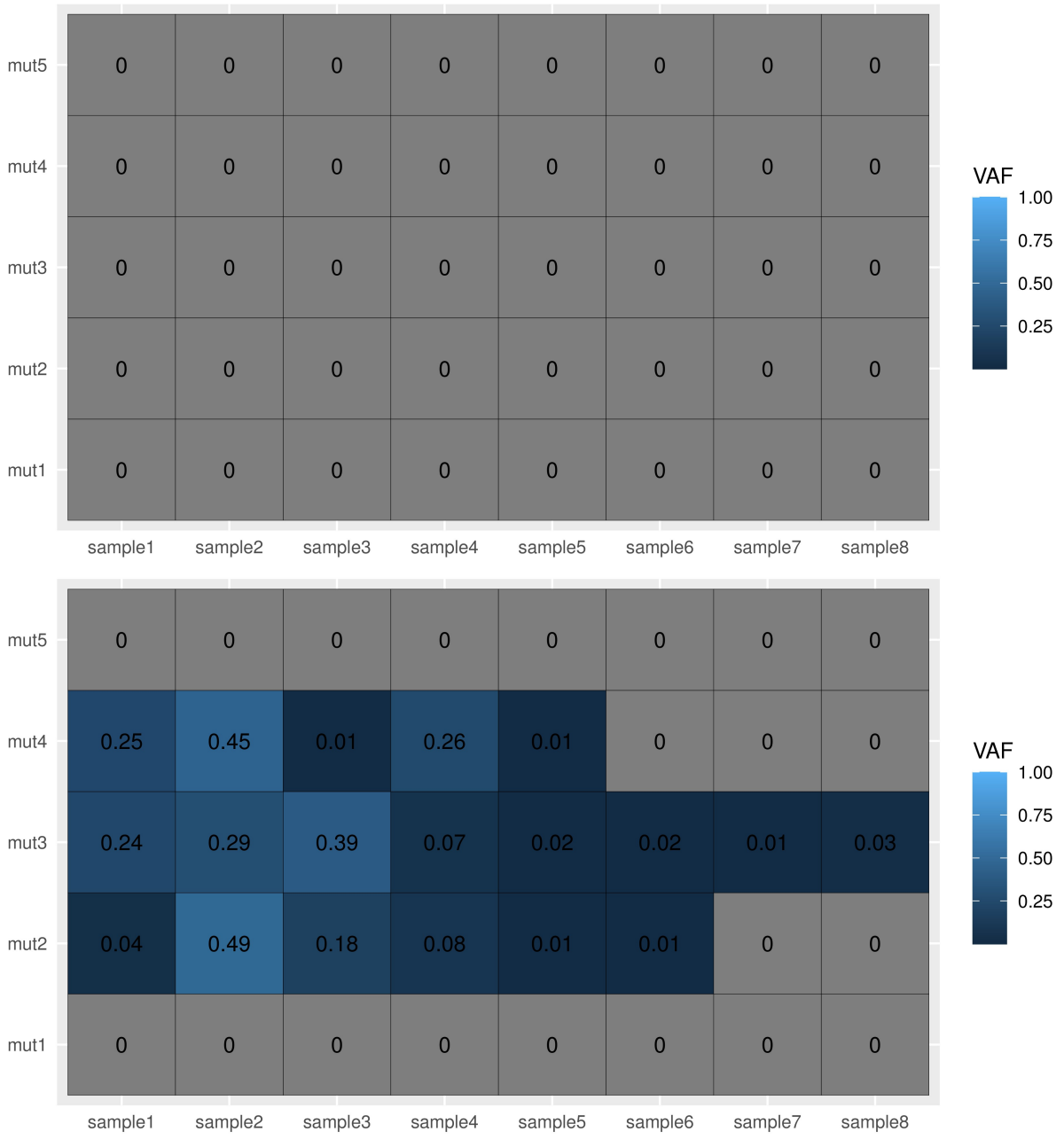
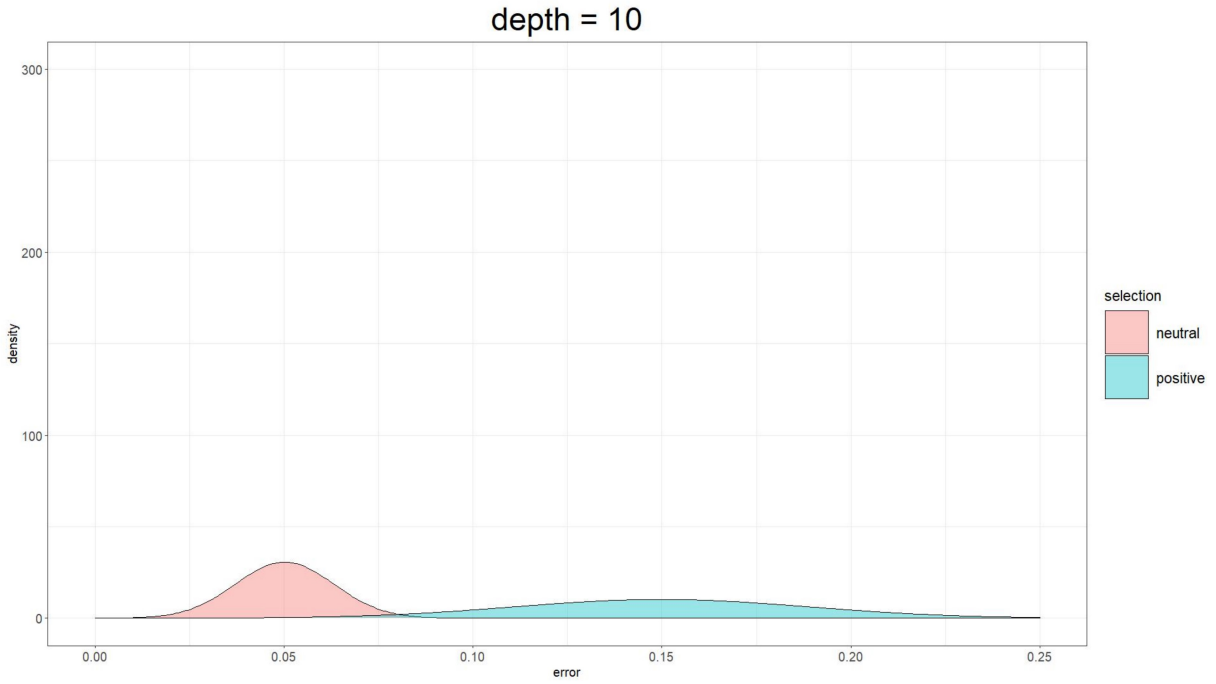
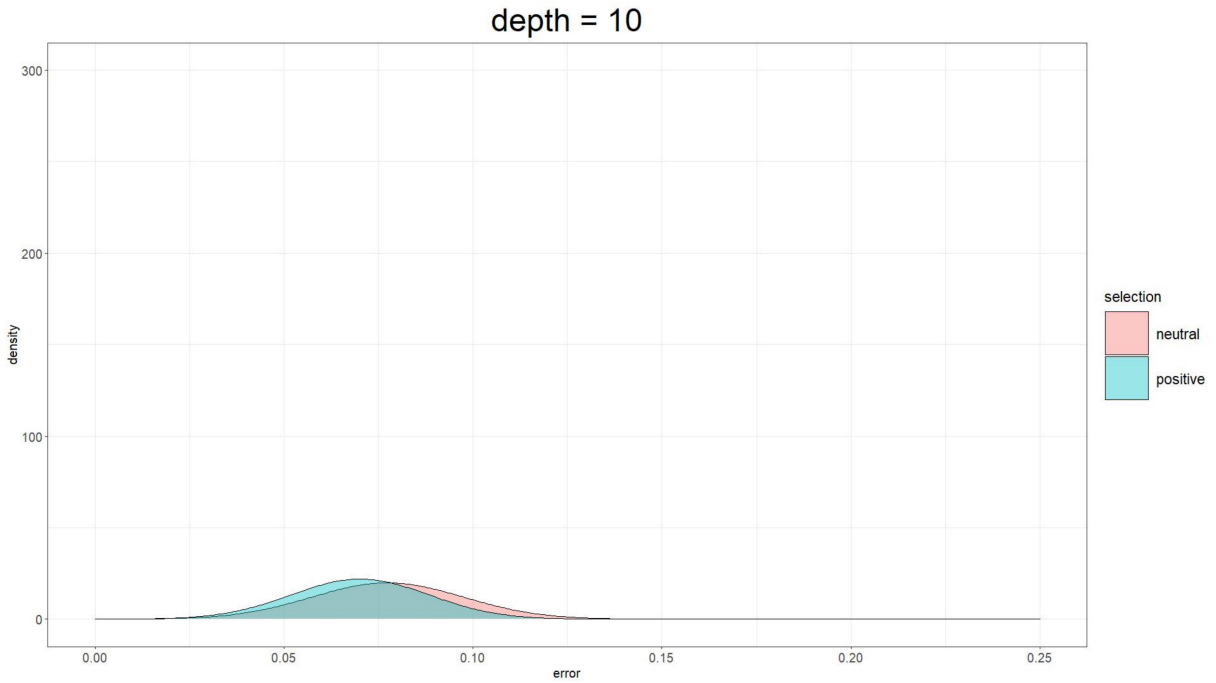


Figure 2: The effect of noise.



The evolution of the error in VAF values based on the read depth sequencing values for a tumor composed by 10 clones and 2 samples.



The evolution of the error in VAF values based on the read depth sequencing values for tumors composed by 100 clones and 10 samples.

It is visible that both instances present a noise-error shrink as depth values get higher, whether they follow a neutral evolutionary selection or a positive-driven one. In addition, the noise-error of bigger instances converges faster than the error of smaller instances. However, the first animation shows that, even if the

smaller instance needs higher values of depth to converge, it ends up producing minor errors.

After analyzing the effect of the different parameters for creating a tumor instance, we will present the basis of the `Phylotree` S4 class.

Step 2: The `Phylotree` S4 class

In this section, we will be using the `Phylotree` class for the purpose of inferring and visualizing phylogenetic trees on the basis of simulated tumor data. The `Phylotree` S4 class is a structure that provides facilities for constructing phylogenetic trees in order to analyze the evolutionary development of tumors. Now, we will show the composition of `Phylotree` class objects and the different techniques for instantiating them.

The structure of `Phylotree` class objects

As every S4 class, the `Phylotree` class is composed by various attributes that are essential for building the phylogenetic tree of a particular tumor instance in an optimal way.

The attributes of `Phylotree` class may be visualized in the following table:

Parameter	Description	Type
B	The square matrix containing the mutations of the clones in the tumor	Matrix
clones	The equivalence table of the clones in the B matrix of the tumor	Vector
genes	The equivalence table of the genes in the tumor	vector
parents	The vector of the parents of the clones in the phylogenetic tree	vector
tree	the `Node` structure that represents the phylogenetic tree	`Node`
labels	The tags of the genes in the phylogenetic tree	vector

Even if `Phylotree` class objects have quite a few attributes, the users of this package will not need to manipulate them as these exist only to reduce the computational cost of the visualization of phylogenetic trees and their comparison.

Once we have explained the structure of `Phylotree` class objects, we will proceed to show the different ways for instantiating them.

The instantiation of `Phylotree` class objects

The mutations of the clone subpopulations in a tumor sample are represented in a n x n binary clone genotype B matrix. Each b_i row of the B matrix represents the mutations in clone v_i. We can analyze the ancestral relationships among the clones of tumors through their B matrices in order to instantiate new `Phylotree` class objects.

Now, we will show an example of the instantiation a `Phylotree` class object on the basis of the B matrix of the simulation of a tumor composed by 5 nodes:

```
# The creation of an instance with 5 clones, 4 samples, K=0.5 and sequencing noise:
instance <- create_instance(n=5, m=4, k=0.5, selection="positive", noisy=TRUE)

# The creation of the Phylotree class object using the previously generated B matrix:
phylotree <- B_to_phylotree(B=instance$B)
```

The *B_to_phylotree* method takes a B matrix as an argument and calculates the values for the other attributes present in `Phylotree` class objects. After instantiating the new `Phylotree` object, we can visualize it using the generic method *plot* for this class:

```
plot(phylotree)
```

Phylogenetic tree composed by 5 nodes.

Nonetheless, this is not the only way for visualizing the phylogenetic tree of a tumor. Note that that this package allows its users to instantiate `Phylotree` class objects using tags for the names of the clones that compose its phylogenetic tree. This is an example about how this can be done:

```
#Create a list with the tags we want to insert to the Phylotree class object
#(the length of the list must be equal to the number of clones in the phylogenetic tree):
tags <- c("mut1", "mut2", "mut3", "mut4", "mut5")

# Create the Phylotree class object inserting the tags we previously created:
phylotree <- B_to_phylotree(B=instance$B, labels=tags)
```

After creating the `Phylotree` class object, we may render it using the tags we have previously assigned to the clones in the following way:

```
plot(phylotree, labels=TRUE)
```

Phylogenetic tree of 5 clone with tags.

This is one of the possible methods for instantiating `Phylotree` class objects on the basis of the B matrix of a tumor. However, this package also grants the option of using the general constructor for the `Phylotree` S4 class for instantiating new `Phylotree` objects, which allows users to give specific values to the attributes of a new `Phylotree` class object.

Once we have shown the usage of the methods for instantiating `Phylotree` class objects and the procedures by these can be visualized, we will proceed to present the functions for comparing the phylogeny of different tumors.

Step 3: Comparing and combining different phylogenetic trees

This package presents different functionalities for comparing the phylogeny of various phylogenetic trees. In order to show how these capabilities work, we will use the $B_mats dataset of the `GeRnika` package, which contains 10 trios of B matrices based on the solution of various instances of the Clonal Deconvolution and Evolution Problem given by the *ILS* and *GRASP* methods. This trios consist of the following matrices:

- **B_true**: The real B matrix of a simulated tumor instance.
- **B_Grasp**: The initial solution of the *ILS* for finding better solutions for the problem instance. This is generated employing a greedy randomized adaptive heuristic strategy.
- **B_opt**: The optimal solution for the instance of the CDEP, obtained from the *ILS*.

First, we will load the real B matrix of the simulated instance of a tumor and the B matrices given by the *ILS* and the *GRASP* method:

```
# Load the predefined B matrices of the package:
B_mats <- GeRnika::B_mats

# Get the B matrices from B_mats for comparing them:
B_real <- B_mats[[2]]$B_real
B_grasp <- B_mats[[2]]$B_grasp
B_opt <- B_mats[[2]]$B_opt

# Create the list of the tags for the clones that compose the phylogenetic trees:
tags <- c("mut1", "mut2", "mut3", "mut4", "mut5", "mut6",
            "mut7", "mut8", "mut9", "mut10")
```

```
# Create the Phylotree class objects using the previously loaded B matrices:
phylotree_real <- B_to_phylotree(B=B_real, labels=tags)
phylotree_grasp <- B_to_phylotree(B=B_grasp, labels=tags)
phylotree_opt <- B_to_phylotree(B=B_opt, labels=tags)


# Render all trees:
plot(phylotree_real)
plot(phylotree_grasp)
plot(phylotree_opt)
```

Visualizing *phylotree_real*, *phylotree_grasp* and *phylotree_opt*

As these three trees above are based on the solution of the same instance for the CDEP, it is reasonable that they are quite similar. Now, we will show the different methods offered by the `GeRnika` package for comparing phylogenetic trees.

The *equals* method

If we compare the phylogenetic trees from above, it is evident that they are not equal. For example, *phylotree_real* and *phylotree_opt* are not equal as some of the edges of *phylotree_real* do not exist in *phylotree_opt* and the other way around.

The equivalence between two phylogenetic trees may be checked by using the *equals* method as follows:

```
# Checking if phylotree_real is equal to itself:
equals(phylotree_1=phylotree_real, phylotree_2=phylotree_real)
#> [1] TRUE

# Checking if phylotree_real and phylotree_opt are equal:
equals(phylotree_1=phylotree_real, phylotree_2=phylotree_opt)
#> [1] FALSE
```

Equal phylogenetic trees, by definition, are composed by the same nodes, connected by the same edges. As a result, this method returns *TRUE* when we compare *phylotree_real* with itself. However, as *phylotree_real* and *phylotree_opt* are not equal, this method returns *FALSE* when we check whether they are equal or not.

Nevertheless, the fact of two phylogenetic trees not being equal does not mean that they do not have commonalities, as they may share common subtrees.

The *find_common_subtrees* method

In order to find the common subtrees between two phylogenetic trees, the following command may be used:

```
find_common_subtrees(phylotree_1=phylotree_real, phylotree_2=phylotree_grasp)
#> Independent edges of tree1:  6
#> Independent edges of tree2:  6
#> Common edges:  3
#> Distance:  12
```

The common subtrees between *phylotree_real* and *phylotree_grasp*.

```
find_common_subtrees(phylotree_1=phylotree_real, phylotree_2=phylotree_opt)
#> Independent edges of tree1:  3
#> Independent edges of tree2:  3
#> Common edges:  6
#> Distance:  6
```

The common subtrees between *phylotree_real* and *phylotree_opt*.

The *find_common_subtrees* function renders all the common subtrees between two phylogenetic trees. For example, the first call of this method shows that *phylotree_real* and *phylotree_opt* have a common subtree that covers the biggest part of both phylogenetic trees. In addition, this method prints the information about the similarities and the distance between both trees.

Furthermore, this method provides the option to render the common subtrees between two phylogenetic trees using the predefined tags for their clones. This can be done in the following way:

```
find_common_subtrees(phylotree_1=phylotree_real, phylotree_2=phylotree_opt, labels=TRUE)
#> Independent edges of tree1:  3
#> Independent edges of tree2:  3
#> Common edges:  6
#> Distance:  6
```

```
find_common_subtrees(phylotree_1=phylotree_real, phylotree_2=phylotree_grasp, labels=TRUE)
#> Independent edges of tree1:  6
#> Independent edges of tree2:  6
#> Common edges:  3
#> Distance:  12
```

The common subtrees between *phylotree_real* and *phylotree_grasp* using tags.

The common subtrees between *phylotree_real* and *phylotree_opt* using tags.

It is perceptible that $phylotree_real$ is more similar to $phylotree_opt$ than to $phylotree_grasp$. As $phylotree_grasp$ represents the initial solution of the Iterated Local Search while $phylotree_opt$ describes the optimal solution after performing the ILS, it is reasonable that $phylotree_opt$ is more similar to $phylotree_true$ than $phylotree_grasp$.

However, this is not the only way for comparing the commonalities between two different phylogenetic trees. It is also possible to combine two phylogenetic trees into a graph that gathers the nodes and the edges of both of them; a consensus tree.

The *combine_trees* method

The `GeRnika` package contains a method for combining different phylogenetic trees, through which their common edges will be distinguished from the independent edges of each of them.

```
# Creating the consensus tree between phylotree_real and phylotree_grasp
consensus_real_grasp <- combine_trees(phylotree_1=phylotree_real, phylotree_2=phylotree_grasp)


# Creating the consensus tree between phylotree_real and phylotree_opt
consensus_real_opt <- combine_trees(phylotree_1=phylotree_real, phylotree_2=phylotree_opt)
```

Now we will render both consensus trees.

```
# Rendering the consensus between phylotree_real and phylotree_grasp
render_graph(consensus_real_grasp)
```

The consensus tree between $phylotree_real$ and $phylotree_grasp$.

```
# Rendering the consensus between phylotree_real and phylotree_opt
render_graph(consensus_real_opt)
```

The consensus tree between *phylotree_real* and *phylotree_opt*.

The above figures present the consensus tree between *phylotree_real* and *phylotree_opt* and the consensus tree between *phylotree_real* and *phylotree_grasp*, respectively. Regarding the trees, the nodes and the

edges that compose the common subtrees between the original trees are blue. In addition, yellow edges denote to the independent edges of the tree passed as the first parameter of the method, while orange edges represent the independent edges of the second tree.

Additionally, `GeRnika` gives users the option to build consensus trees using the tags of the clones that compose the phylogenetic trees. Moreover, it is possible to select the palette of colors in which the edges of the consensus tree will be printed. For this purpose, For this purpose, the `GeRnika` package offers three custom palettes to be used for its methods: "Lancet", "NEJM" and "Simpsons". The "Simpsons" palette is used by default for the methods of `GeRnika`.

```
# Load one of the default palettes of the package:
palette <- GeRnika::palettes$Lancet

# Create the consensus tree between phylotree_real and phylotree_opt
# using clone tags and the previously loaded palette:
consensus <- combine_trees(phylotree_1=phylotree_real, phylotree_2=phylotree_opt,
                           labels=TRUE, palette=palette)

# Render the new consensus phylogenetic tree:
render_graph(consensus)
```

The consensus tree between *phylotree_real* and *phylotree_opt* using tags and a selected color palette.

Note that the parameter *palette* of this method may take a palette –a vector containing the hexadecimal code of various colors– composed by three colors in order to use them for building the consensus tree.

Session information

This is the information of the system on which this document was compiled:

```
#> R version 4.0.3 (2020-10-10)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows 10 x64 (build 19042)
#>
#> Matrix products: default
#>
#> locale:
#> [1] LC_COLLATE=Spanish_Spain.1252  LC_CTYPE=Spanish_Spain.1252
#> [3] LC_MONETARY=Spanish_Spain.1252 LC_NUMERIC=C
#> [5] LC_TIME=Spanish_Spain.1252
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#>  [1] ggpubr_0.4.0         GeRnika_1.0.0        colorspace_2.0-0
#>  [4] knitcitations_1.0.12 markdown_1.1         reshape2_1.4.4
#>  [7] MCMCpack_1.5-0       MASS_7.3-53          coda_0.19-4
#> [10] forcats_0.5.0        stringr_1.4.0        dplyr_1.0.2
#> [13] purrr_0.3.4          readr_1.4.0          tidyr_1.1.2
#> [16] tibble_3.0.4         ggplot2_3.3.3        tidyverse_1.3.0
#> [19] DiagrammeR_1.0.6.1   data.tree_1.0.0
#>
#> loaded via a namespace (and not attached):
#>  [1] mcmc_0.9-7         matrixStats_0.58.0 fs_1.5.0           lubridate_1.7.9.2
#>  [5] RColorBrewer_1.1-2 httr_1.4.2         DiagrammeRsvg_0.1  tools_4.0.3
```

```
#>  [9] backports_1.2.0    R6_2.5.0           DBI_1.1.1          withr_2.3.0
#> [13] tidyselect_1.1.0   curl_4.3           compiler_4.0.3     cli_2.2.0
#> [17] rvest_0.3.6        quantreg_5.85      SparseM_1.81       xml2_1.3.2
#> [21] labeling_0.4.2     scales_1.1.1       digest_0.6.27      foreign_0.8-80
#> [25] rmarkdown_2.8      rio_0.5.27         pkgconfig_2.0.3    htmltools_0.5.0
#> [29] dbplyr_2.0.0       htmlwidgets_1.5.3  rlang_0.4.10       readxl_1.3.1
#> [33] rstudioapi_0.13    farver_2.0.3       visNetwork_2.0.9   generics_0.1.0
#> [37] jsonlite_1.7.2     zip_2.1.1          car_3.0-11         magrittr_2.0.1
#> [41] Matrix_1.2-18      Rcpp_1.0.6         munsell_0.5.0      fansi_0.4.1
#> [45] abind_1.4-5        RefManageR_1.3.0   lifecycle_0.2.0    stringi_1.5.3
#> [49] yaml_2.2.1         carData_3.0-4      plyr_1.8.6         grid_4.0.3
#> [53] crayon_1.3.4       lattice_0.20-41    cowplot_1.1.1      haven_2.3.1
#> [57] hms_0.5.3          knitr_1.30         pillar_1.4.7       ggsignif_0.6.2
#> [61] reprex_0.3.0       glue_1.4.2         evaluate_0.14      V8_3.4.2
#> [65] data.table_1.14.0  modelr_0.1.8       vctrs_0.3.6        MatrixModels_0.5-0
#> [69] cellranger_1.1.0   gtable_0.3.0       assertthat_0.2.1   xfun_0.23
#> [73] openxlsx_4.2.4     broom_0.7.8        rstatix_0.7.0      conquer_1.0.2
#> [77] ellipsis_0.3.1
```

Bibliography

[1] Francesco Marass, Florent Mouliere, Ke Yuan, Nitzan Rosenfeld, and Florian Markowetz. A phylogenetic latent feature model for clonal deconvolution. *The Annals of Applied Statistics*, 10:2377–2404, 2016. See pages 1, 2, and 4.

[2] Rebecca Burrell, Nicholas Mcgranahan, Jiri Bartek, and Charles Swanton. The causes and consequences of genetic heterogeneity in cancer evolution. *Nature*, 501:338–45, 09 2013. See page 1.

[3] Mohammed El-Kebir, Layla Oesper, Hannah Acheson-Field, and Benjamin J. Raphael. Reconstruction of clonal trees and tumor composition from multi-sample sequencing data. *Bioinformatics*, 31(12):i62–i70, 06 2015. See pages 1, 2, and 14.

[4] Maitena Tellaetxe Abete. Metaheuristic algorithms for the clonal deconvolution problem. *ADDI: Institutional Repository*, 2021. See pages 2, 12, 14, and 36.

[5] Matthew A. Myers, Gryte Satas, and Benjamin J. Raphael. Calder: Inferring phylogenetic trees from longitudinal tumor samples. *Cell Systems*, 8(6):514–522, 2019. See page 3.

[6] Jason A. Somarelli, Kathryn E. Ware, Rumen Kostadinov, Jeffrey M. Robinson, Hakima Amri, Mones Abu-Asab, Nicolaas Fourie, Rui Diogo, David Swofford, and Jeffrey P. Townsend. Phylooncology: Understanding cancer through phylogenetic analysis. *Biochimica et Biophysica Acta (BBA) - Reviews on Cancer*, 1867(2):101–108, 2017. Evolutionary principles - heterogeneity in cancer? See page 3.

[7] Sylvia Tippmann. Programming tools: Adventures with r. 2015. See page 3.

[8] Habil Zare, Junfeng Wang, Alex Hu, Kris Weber, Josh Smith, Debbie Nickerson, ChaoZhong Song, Daniela Witten, C. Anthony Blau, and William Stafford Noble. *Inferring Clonal Composition from Multiple Sections of a Breast Cancer*, 07 2014. See page 4.

[9] Christopher A. Miller, Brian S. White, Nathan D. Dees, Malachi Griffith, John S. Welch, Obi L. Griffith, Ravi Vij, Michael H. Tomasson, Timothy A. Graubert, Matthew J. Walter, Matthew J. Ellis, William Schierding, John F. DiPersio, Timothy J. Ley, Elaine R. Mardis, Richard K. Wilson, and Li Ding. *SciClone: Inferring Clonal Architecture and Tracking the Spatial and Temporal Patterns of Tumor Evolution*, 08 2014. See page 4.

[10] Yuchao Jiang, Yu Qiu, Andy J. Minn, and Nancy R. Zhang. *Assessing intratumor heterogeneity and tracking longitudinal and spatial clonal evolutionary history by next-generation sequencing*, 2016. See page 4.

[11] White B S Foltz S M Miller C A Luo J Fields R C Maher C A Dang, H X. *ClonEvol: clonal ordering and visualization in cancer sequencing*, 2017. See page 4.

[12] Facundo Munoz Markus Wamser Pierre Formont Kent Russel Noam Ross Duncan Garmonsway Christoph Glur Russ Hyde, Chris Hammill. *data.tree: General Purpose Hierarchical Data Structure*, 2020. R package version 1.0.0. See page 4.

[13] RStudio Hadley Wickham. *tidyverse: Easily Install and Load the 'Tidyverse'*, 2021. R package version 1.3.1. See page 4.

[14] Richard Iannone. *DiagrammeR: Graph/Network Visualization*, 2020. R package version 1.0.6.1. See page 4.

[15] Jong Hee Park Ghislain Vieilledent Michael Malecki Matthew Blackwell Keith Poole Craig Reed Ben Goodrich Ross Ihaka The R Development Core Team The R Foundation Pierre L'Ecuyer Makoto Matsumoto Takuji Nishimura Andrew D. Martin, Kevin M. Quinn. *MCMCpack: Markov Chain Monte Carlo (MCMC) Package*, 2021. R package version 1.5.0. See page 4.

[16] Hadley Wickham. *reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package*, 2020. R package version 1.4.4. See page 4.

[17] Achim Zeileis, Jason C. Fisher, Kurt Hornik, Ross Ihaka, Claire D. McWhite, Paul Murrell, Reto Stauffer, and Claus O. Wilke. colorspace: A toolbox for manipulating and assessing colors and palettes. *Journal of Statistical Software*, 96(1):1–49, 2020. See page 4.

[18] William H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985. See page 19.

[19] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *Society for industrial and Applied Mathematics*, 18(6):1245–1262, 1989. See page 19.

[20] Jennifer Bryan Hadley Wickham. *R Packages*. O'Reilly Media, 2015. See pages 21, 22, 25, and 26.

[21] Davide Bacciu and Daniele Castellana. Learning Tree Distributions by Hidden Markov Models. *Journal of Machine Learning Research*, 1. See page 46.

[22] Davide Bacciu. Hidden Tree Markov Networks: Deep and Wide Learning for Structured Data. *Symposium Series on Computational Intelligence*, 2017. See page 46.