UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
DIPARTIMENTO DI INFORMATICA, SISTEMISTICA E COMUNICAZIONE
DOTTORATO DI RICERCA IN INFORMATICA - CICLO XXVIII


PH.D. THESIS

# Sampling Ancestral Recombination Graphs and Reconstruction of Phylogenetic Trees for Explaining Evolution

*Author:* Anna Paola CARRIERI

*Supervisors:* Prof. Paola BONIZZONI
Prof. Laxmi PARIDA

*Tutor:* Prof. Lucia POMELLO

November 2015

*To my parents Annamaria and Domenico*

# *Acknowledgements*

This thesis is the result of three years of research efforts. Most of such work would not have been possible without the support of many people that I wish to thank. First of all, my gratitude goes to my two advisors.

I express my special appreciation to my advisor Professor Paola Bonizzoni, who dearly supported me all the time, encouraging and guiding my work in the right direction. Her advices on both research as well as personal issues have been priceless. I also want to thank her for all the invaluable helpful suggestions about how to improve the quality of the final work.

I am deeply grateful to my advisor Professor Laxmi Parida, who shared with me her knowledge and work, who stimulated my scientific research allowing my professional growth. Many thanks to Laxmi for giving me the opportunity to work with her and join her team, as visiting research student at IBM T.J. Watson research center, making me feel part of it.

The members of the computational genomics group have contributed immensely to my personal and professional time at IBM. First of all, many thanks to Filippo Utro for always being there ready to answer my questions and clarify my doubts. Thanks to him also for reading the resulting dissertation and giving me insightful suggestions on how to improve my draft. Thanks to Daniel Platt for all the math he wholeheartedly and patiently taught me. Thanks to Niina Haiminen for her friendly and lovely help anytime I needed it.

I express my gratitude to Riccardo Dondi. His fruitful interaction within my research work stimulated and made possible the development of new contributions that are part of this thesis. Thanks to Stefano Beretta and Yuri Pirola for the stimulating and enlightening discussions. Stefano, Riccardo and Yuri constantly helped me and made pleasant, at the same time, the working hours in the lab.

I would like also to thank Professor Lucia Pomello and Professor Gianluca Della Vedova for the brilliant comments and advices they kindly gave me in the past three years.

Last but not least, I would like to thank my parents. They always supported me and pushed me for higher education. I hope they will be proud.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The thesis focuses on computational problems motivated by the need of understanding the evolution of genomic information starting from data produced by Next Generation Sequencing (NGS) technologies. Recent improvements in NGS technologies have drastically reduced the cost of sequencing a whole genome and have led to a huge increase in the amount of DNA/RNA and protein sequences available for the analyses.

Therefore, nowadays there is the necessity of developing new computational models and methods that can manage and elaborate this large collection of sequences efficiently. These data represent a quite useful source of information and pose new computational issues in studying and reconstructing the evolutionary history of genomic sequences in the context of population genomics and comparative genomics. In this dissertation we mainly address the problem of reconstructing evolutionary histories following two research directions: the stochastic simulation of complex scenarios of multiple population evolution with admixture and the combinatorial reconstruction of phylogenetic trees from binary data. Both research directions explore algorithms for the generation (or reconstruction) of graphs (or trees) that model evolution in presence of evolutionary events, such as recombinations and Single Nucleotide Polymorphisms (SNPs). The basic combinatorial models, that are investigated here, are ancestral recombination graphs and phylogenetic trees.

The first research direction regards the design and the development of accurate and efficient algorithms for simulating complex scenarios of multiple population evolution with admixture, which summarize the series of genetic events (such as coalescences, recombinations, mutations). This is an important task for answering many basic questions related to population genomics and several aspects have been studied extensively in the past few decades. Given the large empirical data sets available for the analysis, it is not feasible to model every detail of all the genetic events occurring among genomic

sequences during the evolution. The aim of simulations is not only to capture the resulting extant population samples, for testing complex hypotheses of the effect on the genetic profile of extant populations, but also their relevant evolutionary history. The common evolutionary history of extant samples of a population is captured by a graph, called Ancestral Recombination Graph (ARG), which is an additional important vehicle in hypothesis checking and reconstruction studies. More precisely, an ARG represents genetic exchange events (coalescences and recombinations) that occurs among individuals in the evolving population together with the polymorphisms of the duplication model.

In literature, most models are based on rather simplistic hypothesis of the possible inter-evolution of present day populations. One of the main bottlenecks has been the sheer size of the monolithic common history of multi-populations, each of realistic size. Under these conditions simulators often do not terminate in reasonable time in spite of meaningful parameter settings, even when simulating simple scenarios with just three populations. Therefore, we address the task of developing an efficient backward simulation algorithm for sampling ARGs representing evolutionary histories of several present-day populations that are related and admixed. The inferential analysis is retrospective, i.e., genomic sequences (such as chromosomes or portions of chromosomes) are collected from one or several contemporary populations of a single species and the goal is to understand aspects of the population evolutionary past through analysis of present day samples. This problem becomes more relevant and interesting as more detailed genomes of different organisms, highlighting the unexpected diversity within a species, are available.

Our main contribution, in this context, has been the introduction of a framework for modeling complex evolutionary scenarios and a backward simulation algorithm, named SimRA (Simulation based on Random-graphs Algorithms), that is both time and space efficient enough to be practical [1]. SimRA makes it possible to run hundreds of experiments in very short time, enabling a very effective vehicle of carrying out complex studies. In [1, 2] we show, through extensive experimental analysis, that SimRA has better space and time performance than another established backward simulation algorithm, that we call here Hudson algorithm, and that is the basis of all the backward simulators. Moreover, to the best of our knowledge, in [1], we derive for the first time, analytic forms of the expected ARG characteristics of a population, i.e., height of the graph, number of recombinations, number of mutations and population diversity in terms of its defining parameters. Through simulations we illustrate that the expected values for the ARGs characteristics closely match the empirical values, proving the accuracy of SimRA. Hence, we demonstrate that SimRA is efficient, without compromising any accuracy of the resulting simulations, all the while producing ARGs in compact form. Complex simulation of scenarios results in complex interplay of parameters. Thus, the closed-form functions for ARG characteristics are also crucial in aiding the user to specify

meaningful parameters for simulating complicated scenarios, not through trial-and-error based on raw compute power, but through intelligent parameter estimation.

Moreover, we report how the efficiency of SimRA has been fundamental to conduct controlled simulations with the purpose of validating a new topological framework that allows to detect admixture in related contemporary populations. Relatedness of populations is an interesting problem and has been studied extensively in the population genomics community, for instance in the context of humans and plants. In particular, in plant breeding this understanding is very important in gauging the diversity in the genetic pool and using it effectively in breeding programs. In the context of humans, admixture mapping of the genome is useful for disease or complex trait association studies. In [3] we present the first combinatorial approach, based on persistency in topology, that characterizes admixture in populations. We show, based on controlled simulations computed by SimRA, that the topological characteristics have the potential for detecting subtle admixture in related contemporary populations. Then we apply the technique successfully to a set of avocado germplasm data indicating that the approach can lead to novel characterizations of relatedness in populations.

The second research direction regards the development of efficient algorithms to reconstruct phylogenesis from binary data encoding genomic information obtained from the analysis of NGS genome data. A phylogeny is a tree that models the ancestral relationships among extant taxa or species. We focus on characters-based methods to reconstruct the phylogenesis of contemporary species. A character-based phylogeny is a tree explaining the evolutionary history of a set of species described by genomic characters or attributes, such as haplotypes, protein domains or markers in tumors. Therefore, from a computational point of view, it is not relevant whether we are actually studying taxa or individuals or other genomic data. We will follow the usual convention of calling species the units under study. The problem of reconstructing character-based phylogenesis takes as input a matrix, where the rows are the extant species and the columns are the characters describing the species. Indeed, each species is explained by a set of characters assuming a finite set of states. The goal is to find a phylogeny where the present-day and known species are the leaves, and the internal nodes are labeled just as the leaves by a set of character states. Characters may mutate their states in the tree representing their gain and loss during the evolution. For each edge $(x, y)$ of the phylogeny, the mutated characters along the edge are those whose states for some characters are different in species $x$ and $y$. The simplest case is when all characters are binary, modeling the situation when each species has or does not have a given feature, hence we focus on binary matrices. Character-based methods are based on the maximum parsimony principle that looks for the evolutionary tree explaining the observed characters in actual species with the minimum number of changes (gains

and losses). Established maximum parsimony models (or simply parsimony models) are Dollo and Camin-Sokal, both leading to NP-hard reconstruction problems. On the other hand, the most restrictive maximum parsimony model is the perfect phylogeny, which is based on the infinite site assumption (i.e., each character can be gained at most once in the whole tree). The perfect phylogeny has very efficient polynomial time algorithmic solutions, including linear-time algorithms. This model has been widely investigated and applied in different areas of computational biology specially in the context of haplotype inference. However, the binary perfect phylogeny model is often too restrictive to explain the evolution of real biological data where homoplasy is present. Indeed the perfect phylogeny does not allow to represent homoplasy events, such as recurrent mutations (multiple gains of a character) and back mutations (losses of a character) [4]. Two cases, where those limitations are evident, are the study of carcinogenesis and protein domains evolution. Hence, a central goal is to define new models that are more widely applicable than the perfect phylogeny, while retaining its computational efficiency where possible.

Therefore, we address the problem of reconstructing a variant of the perfect phylogeny model, the persistent perfect phylogeny, which relaxes the strict assumption of the perfect phylogeny, allowing one back mutation for each character. With the aim of investigating its computational complexity, we introduce the Constrained Persistent Perfect Phylogeny problem (CPPP) [5] which generalizes the Persistent Perfect Phylogeny (PPP) problem, by adding constraints for some characters in the input matrix. More precisely, the CPPP problem requires that for some pairs (character, species) neither the species nor any of its ancestors can have the character. We explore some algorithmic solutions for the CPPP problem, in particular providing a polynomial time algorithm for a particular class of input matrices for both the PPP and CPPP problems. Using this result, we also develop a parameterized algorithm for solving the general constrained and unconstrained reconstruction problems where the parameter is the number of characters. We test our algorithm both on simulated and real data. More precisely, we use haplotype data from the set ASW (African ancestry in Southwest USA) coming from the International HapMap project. A preliminary experimental analysis shows that our algorithm performs efficiently on simulated matrices as well as on real data. Indeed, the tree technique adopted, combined with the use of constraints, allows to obtain solutions efficiently for matrices that otherwise would require exponential time. Consequently, the constrained persistent perfect phylogeny model allows to explain efficiently data that do not conform with the classical perfect phylogeny model. Moreover, we identify a family of binary matrices that cannot be explained by the persistent perfect phylogeny model, while all their sub-matrices can [6].

We conclude the thesis presenting results concerning the scaffold filling computational problem. The latter problem derives from the necessity of filling incomplete genomic

sequences, also called scaffolds, with missing genes, in order to reconstruct complete genomes that share a high level of similarities with a known reference genome. This is an intriguing challenge as most of the released genomes, produced by high-throughput NGS technologies, are unfinished and incomplete. These incomplete genomic sequences may introduce errors when used for phylogenetic reconstruction studies and comparative genomic analyses.

Hence, we consider two combinatorial problems: One-sided scaffold filling problem and Two-sided scaffold filling problem. The first problem consists of filling a scaffold $B$ with missing genes so that the resulting complete genome $B'$ maximize the similarity with the reference genome $A$. In the Two-sided case, instead, both genomes are incomplete and the problem consists of filling $A$ and $B$ maximizing the similarity between them. If we consider the maximum number of common adjacencies between two genomes as similarity measure, both One-sided and Two-sided scaffold filling problems are NP-hard under this measure. Hence, a natural goal is the investigation of their parameterized complexity, where the parameter is the maximum number of common adjacencies. Our contribution, in this context, is the development of two Fixed Parameter Tractable (FPT)-algorithms for both problems [7, 8]. The two FPT-algorithms we propose are mainly based on the color-coding technique and a perfect family of hash functions. Moreover, both algorithms find a solution applying dynamic programming steps.

## Thesis structure

The thesis is organized as follows.

In Chapter 2 we present the theoretical basis and some key concepts needed for a better understanding of the results obtained while investigating the first research direction, that is the development of stochastic algorithms for simulating the evolution of populations.

Chapter 3 is devoted to a detailed description of our backward simulation algorithm, called SimRA, that simulates generic and complex scenarios of multiple population evolution with admixture. In this chapter we also present all theoretical contributions and experimentations related to SimRA.

In Chapter 4 we present a new topological framework to detect admixture in related contemporary populations. We show, based on controlled simulations computed by SimRA, that the topological characteristics have the potential for detecting admixture in related present-day populations and we applied the topological framework to real biological data.

Chapter 5 is dedicated to the state of art regarding computational model and methods for phylogenetic reconstruction, useful for a better understanding of the results obtained while investigating this research direction. More precisely, we present a survey of character based methods to reconstruct phylogenesis and we focus our attention on the perfect phylogeny model and one of its variant, called persistent perfect phylogeny.

In Chapter 6 we introduce the Constrained Persistent Perfect Phylogeny (CPPP) problem and we present a polynomial time algorithm for a particular class of instances and a parameterized algorithm for the general CPPP and PPP problems. We also give a characterization of binary matrices that cannot be described by the persistent perfect phylogeny.

Chapter 7 describes two Fixed-Parameter Tractable algorithms for the One-sided scaffold filling and Two-sided scaffold filling problems respectively.

Finally, in Chapter 8 we summarize our contributions and we present interesting future directions and goals that can be investigated.

# Chapter 2

# The coalescent theory and the ancestral recombination graph

In this chapter we present some key concepts useful for a better understanding of the results obtained while investigating the first research direction, that is the development of stochastic algorithms for simulating complex scenarios of multiple population evolution.

More precisely, we give a brief overview of forward and backward simulation tools and we introduce the *coalescent model*, a mathematical model for common evolutionary history of a set of genomic sequences or chromosomes. Moreover, we describe the established Write-Fisher model of evolving population (which is the basis of the discrete and the continuous coalescent processes for a sample of genomic sequences) and we explain how it can be extended to include recombination. We present a fundamental mathematical object, the Ancestral Recombination Graph (ARG), modeling the historical genetic exchange events that occur among genomic sequences. Finally, we report a basic and well-known backward simulation algorithm, that we call Hudson algorithm, for sampling an ARG of a population of individuals (genomic sequences). The Write-Fisher model with recombination, the ARG and Hudson algorithm are the theoretical basis of the framework for modeling complex evolutionary scenarios and the backward simulation algorithm proposed in Chapter 3.

## 2.1 Forward and backward simulation algorithms

In population genomics, simulation algorithms are useful to understand the evolutionary consequences of complex systems, that are populations of individuals where interactions among these are difficult to predict analitically [9]. In literature many simulation tools

have been presented. The interested reader is referred to [9] for a wide review of them. However, in the following we want to classify simulators based on their underlying approaches. More precisely, there are two categories of simulation systems, forward and backward, that differ in capabilities and computation times, hence they are suitable for addressing different questions. The main output for simulators is a sample of $m$ genetic sequences, given the population of size $N$ along with other parameters. In both approaches a genealogical structure is constructed. Indeed, the common history of a population is represented by a genealogy[1] where the leaves are all homologous copies of the same genetic segment in the genome and are collected from one or several contemporary populations of a single species [10]. A population is best understood as a population of $N$ genomic sequences, that are genes or genomes or chromosomes, rather than a population of individuals. A genomic sequence is composed by nucleotides and two sequences of the same genetic region are different if they do not have identical alleles. A lineage is a line of descent or ancestry for a homologous genomic sequence or a locus (regardless of whether or not copies of the locus are identical or different).

### 2.1.1 Forward simulators

Forward simulation is conceptually the simpler of the two approaches, indeed events proceeds forward in time, that is from past to present. Forwards-in-time simulators are centred on individuals: each individual in the simulated population (or populations) follows a life cycle (i.e., birth, selection, mating, reproduction, mutation, migration and death) [9]. Looking forward, given the known individuals at each generation in the pedigree[2], the relationships between ancestors and descendants are traced forward in time to the most recent generation by means of basic probabilities [11]. This approach allows researchers to monitor changes in the genetic landscape of a population analyzing samples at specific time intervals. Moreover, an advantage of this approach is its adaptability to diverse evolutionary forces, such as genetic drift, selection and mutation. For instance, SFS_CODE [12] is a forward simulator that allows to handle effects of migration, demographics, and selection. Fregene [13] incorporates selection, recombination (crossovers and gene conversion), population size and structure, and migration. simuPOP [14] is an individual-based forward simulation environment. This system implements interactive evolution of populations and allows to model complex evolutionary scenarios in the environment [15].

[1] A genealogy, also known as family history, is the record of ancestor–descendant relationships (the trace of lineages) for a family or population.

[2] The pedigree is the complete genealogical network representing the evolution of a population.

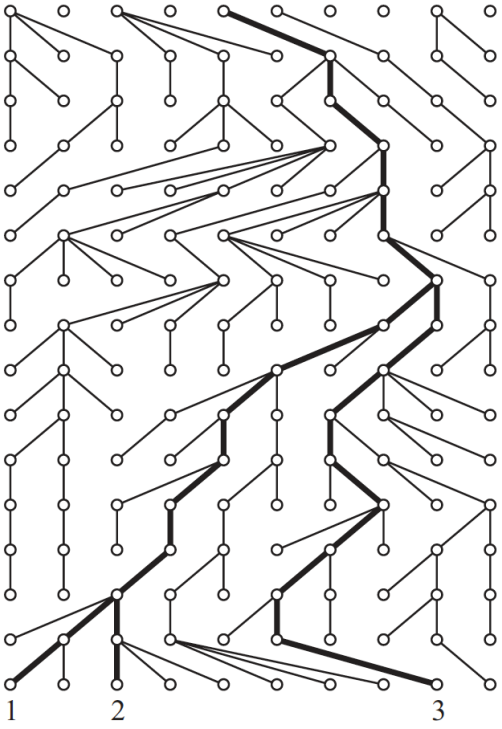



FIGURE 2.1: The genealogy of three randomly sampled sequences (1, 2, 3) from left to right. Edges tracking backward the ancestors of these three sequences are highlighted. Two generations back in time, sequences 1 and 2, find a common ancestor and this lineage might be labelled (1, 2) to reflect this fact. The ancestry of the sequences are marked by bold lines sixteen generations back in time. Nine generations back all three sampled sequences have found a common ancestor. The Figure has been taken from Chapter 1 of [10].

## 2.2 The coalescent model and the backward approach

While a natural direction to proceed is forward in time, another type of analysis of ancestor–descendant relationships is possible based on a retrospective or backward approach from the present time to the past. In principle, this is more economical in space and time as we will discuss more in details in Section 2.8. The underling mathematically approach consists of simulating the time to the next genetic event back in the past without explicitly simulating every generation. Indeed, this backward approach is the base of a powerful set of models known as coalescent theory.

In the following, we describe the coalescent model that captures the backward approach. The prospective of the coalescent model is to predict the probability that two lineages trace back in time to a single ancestral lineage in a previous generation. This event is called *coalescence* or *coalescent event* and literally means to grow together or to fuse. A central concept in coalescent theory is connecting a group of extant lineages back through time to a single ancestor in the past. The latter is the first ancestor (going backward

in time) of all the lineages in a sample at the present time and is referred as the most recent common ancestor or MRCA (See Fig. 2.1 as an example).

The inferential analysis is retrospective, meaning that we want to infer details about the sample's (and the population's) evolutionary past through analysis of the present day sample [10]. More precisely, given a collection of $N$ chromosomal segments (a population), under the best computational scenario (in terms of time, space and sophistication of algorithms), the aim consists of constructing the evolutionary history (the genealogy) of these $N$ extant sequences by means of *simulation algorithms*.

Simulation algorithms are based on a generative coalescent model that simulates the population by evolving it over time. We consider the Wright-Fisher [16, 17] model of evolving population, since it is one of the most applied and well-known.

## 2.3 The Wright-Fisher Model of Evolving Population

Wright [16] and Fisher [17] introduced a simple model of evolving population that describes the genealogical relationships of genetic sequences or genomes. This basic model of reproduction provides a dynamic description of the evolution of an idealized population and the transmission of genetic sequences from one generation to the next one.

Genealogies in the diploid model and the haploid model are probabilistically similar for large choices of effective population size $N$. However, for convenience we consider an haploid population of $N$ individuals. In the haploid model, all individuals choose independently of each other. In other words, each individual of generation $j$ choose one random parent from the previous generation $j + 1$. The generations are discrete, i.e., starting from the present time, or generation 0, we go back to the past considering previous generation 1, 2, and so on.

The Wright-Fisher Model assumes some important properties of the evolving population [10]:

1. *discrete and not overlapping generations*: in the case of humans, this is equivalent to assuming that everybody has the same lifetime expectancy from conception to reproduction, and that reproduction and death are simultaneous and synchronous for all individuals;

2. *constant population size*: observe that the model would be different if the population is growing, contracting, oscillating;

3. *panmictic population*: all individuals have equal ration and are equally fit. Indeed, panmixia means random mating, i.e., the mating between two organisms is not influenced by any environmental, hereditary, or social interaction. A panmictic population is one where all individuals are potential partners. This assumption is convenient for the basic theoretical concepts, but presumably not realistic. Indeed it can be relaxed to investigate the presence and strength of natural selection.

## 2.4 The discrete-time coalescent process

Using rules of random sampling based on the assumptions of the Wright–Fisher model of reproduction and the properties of the associated probability distributions (the binomial, the geometric and the exponential distributions) we can derive the basic coalescent process [10]. The original formulation of the coalescent process has been given by Kingman in 1982 [18], termed as the *n-coalescent* or just the *coalescent* for a sample of $m$ genomic sequences.

Again, we consider the haploid model since is more straightforward and predictions that follow from the haploid coalescent model can be applied to samples of lineages from diploid organisms. Moreover, the coalescent sampling process can be extended to approximate the process of reproduction for diploid lineages.

The constant population size is $N$. Consider a random sample of two lineages of the $N$ total lineages in the present generation. We can develop a prediction for the number of generations back in time until two lineages coalesce to a single lineage. Given that one of these two sampled lineages can choose its ancestor freely in the previous generation, for coalescence to occur, the second lineage must choose the same ancestor as the first lineage, which is one out of $N$ possible ancestors in the previous generation. Hence, the probability of coalescence for two lineages is $\frac{1}{N}$ whereas the probability that two lineages do not have a common ancestor in the previous generation is $1 - \frac{1}{N}$.

Observe that the probability of coalescence for two lineages depends only on the current generation. In other words, sampling in different generations is independent of each other. Hence, the probability that two randomly sampled lineages coalesce to their common ancestor back in time at an arbitrary generation $j$ is

$$P(T_c^2 = j) = \left(1 - \frac{1}{N}\right)^{j-1} \frac{1}{N}, \tag{2.1}$$

where $T_c^2$ denote the time of coalescent event back in the past. In the first $j-1$ generations they chose different ancestors, and then in generation $j$ they chose the same ancestor.

In the coalescent process the average time to a coalescent event is called *waiting time* and it is simply the inverse of the probability of a coalesce event. Thus, the probability for a pair of lineages to occur is $\frac{1}{N}$, then the waiting time is $N$.

Suppose we want to determine the waiting time for $k$ lineages, where $k$ is less than $m$ (total number of lineages sampled from a population of $N$). When no coalescence event occurs, one lineage finds its ancestor among any of $N$ individuals in the previous generation, meaning that the next lineage must find its ancestor among $N-1$ individuals over $N$ in the previous generation, and again the next lineage has to find its ancestor among $N-2$ individuals and so on till the final lineage must find its ancestor among $N-(k-1)$ possible parents.

Hence, the probability that $k$ lineage over $N$ do not coalesce (i.e., $k$ lineages have $k$ different ancestors in the previous generation) is:

$$1\left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right)\left(\frac{N-3}{N}\right)\dots\left(\frac{N-(k-1)}{N}\right)$$

that can also be written as:

$$1\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\left(1-\frac{3}{N}\right)\dots\left(1-\frac{k-1}{N}\right)=\prod_{i=1}^{k-1}(1-\frac{i}{N}). \tag{2.2}$$

If the number $k$ of lineages sampled is much smaller than the total number of lineages in the population $N$ then the probability of non-coalescence for $k$ lineages can be approximated by

$$1-\binom{k}{2}\frac{1}{N}, \tag{2.3}$$

where the binomial factor $\binom{k}{2}$ enumerates all different ways to uniquely sample pairs of lineages from a total of $k$ that can be also written as:

$$\binom{k}{2}=\frac{k(k-1)}{2}. \tag{2.4}$$

On the other hand, the probability of coalescence for any of the unique pairs among $k$ lineages is

$$\binom{k}{2}\frac{1}{N}=\frac{k(k-1)}{2N}. \tag{2.5}$$

Consequently, the probability that $k$ lineages experience a single coalescence event at generation $j$ (i.e., 2 lineages out of $k$ find a common ancestor at an arbitrary generation $T_c^k = j$ back in the past, for $j = 1, 2 \dots$ ) is:

$$P(T_c^k = j) \approx \left(1 - \frac{k(k-1)}{2N}\right)^{j-1} \left(\frac{k(k-1)}{2N}\right), \tag{2.6}$$

that can be written also as:

$$P(T_c^k = j) \approx \left(1 - \binom{k}{2}\frac{1}{N}\right)^{j-1} \binom{k}{2}\frac{1}{N}. \tag{2.7}$$

$T_c^k$ has approximately a geometric distribution with parameter $\binom{k}{2}\frac{1}{N}$ . Since all pairs of genes are equally likely to find a common ancestor, the pair that finds a common ancestor is chosen with equal probability among the $\binom{k}{2}$. The times of coalescence are independent [10].

## 2.5 The continuous-time coalescent process

In the Wright–Fisher model time is measured in generations that are discrete units. However, from a computational and conceptual point of view it is convenient to consider continuous time approximations [10]. Indeed, the series of failures (non-coalescence) until a success (coalescence) in the genealogical process can be modeled where time is continuous.

Let $j$ be the time measured in generations, to derive the continuous coalescent process we set the continuous time $t = \frac{j}{N}$. It follows that $j = Nt$ is the continuous time $t$ back into $j$ generations.

The probability of coalescence at time $t$ is approximated using an exponential distribution.

### 2.5.1 The exponential distribution

The exponential distribution describes situations in which an object initially in one state can change to an alternative state with some probability that remains constant through time.

Recall that the cumulative distribution function, CDF, of an exponential distribution with parameter $\lambda$ is

$$\text{Exp}(\lambda) = \left[P_{T \leqslant t} = 1 - e^{-\lambda t}\right]. \tag{2.8}$$

Consider a discrete distribution (where $j$ is an integer) and parameter $\lambda' < 1$ as

$$P_{J \leqslant j} = 1 - (1 - \lambda')^j. \tag{2.9}$$

For some scaling factor $N$ as $j = Nt$ the two distributions are related as follows:

$$\left[P_{T \leqslant t} = 1 - \left(1 - \frac{\lambda}{N}\right)^{Nt}\right] \quad \longrightarrow \quad \left[P_{T \leqslant t} = 1 - e^{-\lambda t}\right], \tag{2.10}$$

as $N \longrightarrow \infty$. In fact the above holds for small $p$ and large $N$ with $\lambda = pN$.

Note that, assuming that the effective population size $N$ is large , then the probability of coalescence is small $\frac{1}{N}$.

The mean of the exponential distribution is

$$E(x = t) = \frac{1}{\lambda}$$

and the variance is:

$$E(x) = \frac{1}{\lambda^2}$$

The following properties of an exponential distribution are used to deal with multiple exponential distributions, each with possibly different parameters.

Let $X \sim \text{Exp}(\lambda_1)$ and $Y \sim \text{Exp}(\lambda_2)$. Then:

**Property 1.**

$$\min(X, Y) \sim Exp(\lambda_1 + \lambda_2).$$

**Property 2.**

$$P(X < Y) = \frac{\lambda_1}{\lambda_1 + \lambda_2}.$$

**Property 3.**

$$\mathbb{E}(X) = 1/\lambda_1.$$

The exponential distribution can be used to obtain an approximate average and variance for coalescence times. In other words, the cumulative exponential distribution is used to approximate the probability that a coalescent event occurs at or before some time $t$. The waiting time, $T_k^c$, in the continuous representation for $k$ genes to have $k-1$ ancestors is exponentially distributed with $T_k^c \approx \exp(\binom{k}{2})$, that is:

$$P(T_k^c \leqslant t) \approx 1 - e^{-tN\frac{k(k-1)}{2N}} = 1 - e^{-t\frac{k(k-1)}{2}} \tag{2.11}$$

Recall that $\frac{k(k-1)}{2} = \binom{k}{2}$. This is derived from equations (2.5) and (2.6). where $T_k^c$ is the time to coalescence e $t$ is scaled in unit of $N$ generations, i.e., $t = \frac{j}{N}$. Note that the probability of coalescence increase more rapidly toward one for larger numbers of lineages $j$.

In the following, we describe a stochastic algorithm that samples genealogies for $m$ genomic extant sequences [10].

**Algorithm**

1. Start with $k = m$ active lineages;
2. Simulate the waiting time $T_k^c$ to the closest coalescent event back in the past, i.e., $T_k^c = \exp(\binom{k}{2})$;
3. Choose a random pair $(i, j)$ of active lineages uniformly among the $\binom{k}{2}$ possible pairs;
4. Merge $i$ and $j$ into a new active lineage and decrease $k$ by one, $k \to k - 1$;
5. If $k > 1$ go to 2, otherwise stop;

## 2.6 Modeling recombination in the coalescent model

The coalescent model has been also extended to include *recombination*, that is one of the primary genetic events shaping an autosomal chromosome. This genetic exchange event consists of the production of offspring having different combinations of homologous chromosomal segments, that differ from those found in either parent. In eukaryotes, recombination naturally occurs during meiosis and leads to a novel set of genetic information that can be passed on from the parents to the offspring. Recombination complicates the genetic landscape of a population, and understanding the manifestations of this genetic exchange event in the evolutionary history of chromosome sequences has been a subject of intense studies [19]. In case of recombinations the structure needed to describe the relationships of a set of genomics sequences is a complicated graph rather than a single tree.

Hudson introduced recombination into the coalescent process and presented a simple model in 1983 [20]. Hudson's model of recombination is illustrated in Fig. 2.2.

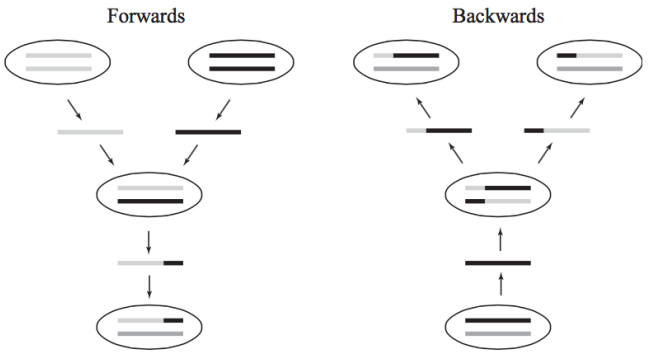



FIGURE 2.2: This Figure shows the Hudson model of recombination from a backward and a forward prospective. Looking forward, the allelic states of the grandparent's chromosomes determine one of the child's two chromosomes. Looking backward, the child's dark grey chromosome is inherited through the other parent and the dark grey chromosomes in grandparents have unknown allelic states. The Figure has been taken from Chapter 5 of [10].

When recombination occurs, a random point uniformly along the chromosome is picked. Then the genetic material to the left of this point from one parent chromosome is copied and the genetic material to the right of this point from the other parent chromosome is copied. Looking forward, two sequences are recombined into one recombinant sequence. Looking backward, an individual chooses a chromosome from one parent and this chromosome is split into two grandparental chromosomes.

Note that recombination events are the opposite of coalescent events that, instead, combine two sampled sequences into one ancestor. Indeed, from a backward prospective in time, recombination causes splitting and coalescent causes the merge of sequences.

In [20] the coalescent process with recombination is formulated as competing exponentially distributed, and independent, waiting times for coalescent and recombination events. In other words, recombination and coalescent are competing processes that determine the genealogy of the sample. The latter is a graph rather than a tree. The parameter of the exponential distribution, determining the coalescent intensity, depends on the number of ancestors carrying ancestral material to the sample. On the other hand, the parameter of the exponential distribution for the intensity of recombination depends on the scaled recombination rate $\rho$ over the sequence times the number of ancestral lineages. The scaled recombination rate is $\rho = Nr$, where $r$ is the recombination rate, i.e., the expected number of recombination events in the population of size $N$ in one generation.

Once these intensities have been specified, a backward algorithm for simulating the process includes splitting (recombination) and joining (coalescence) of ancestors until a single ancestor is produced, called the Grand Most Recent Common Ancestor (GMRCA). The resulting structure is a graph, called Ancestral Recombination Graph (ARG).

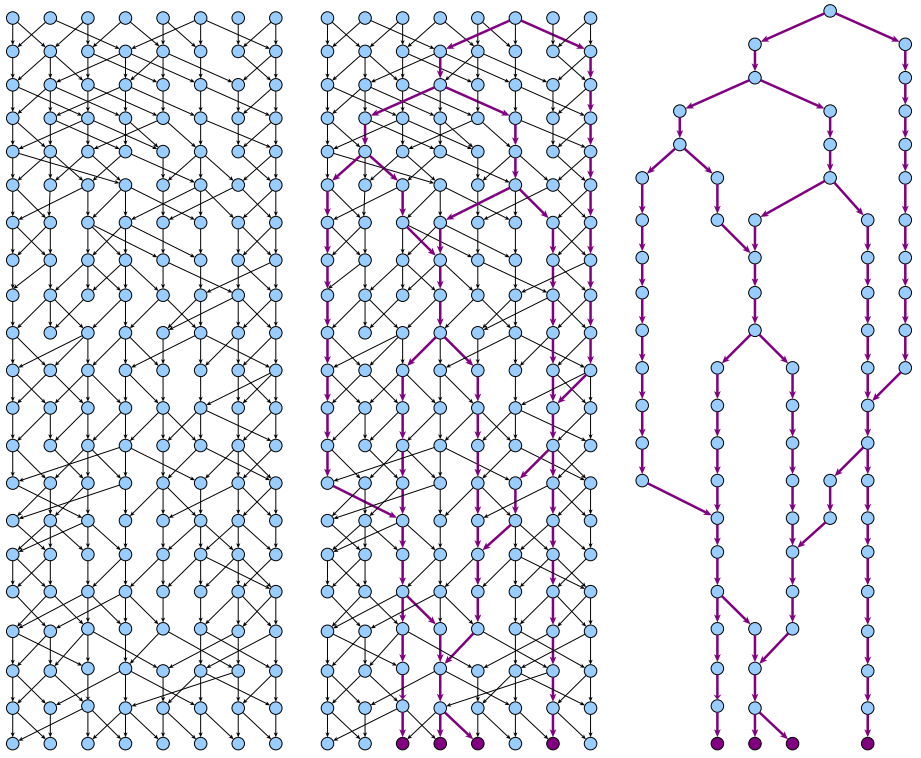



FIGURE 2.3: (a) shows the complete genealogical (pedigree) graph of a Wright Fisher population of 8 individuals at each generation. Every individual has exactly 2 parents. (b) shows the substructure of (a) based on tracking some chromosomal segment from 4 extant samples, marked in dark purple. The bold edges mark the flow of the genetic segments of interest to the 4 extant units. (c) shows the relevant part, the ARG, by removing the extraneous parts of the network of (b). Note that a forward simulator (moving in time from past to present) may have to simulate the network in (a) or (b), whereas the network in (c) is adequate for a backward simulator (moving in time from present to past).

## 2.7 The Ancestral Recombination Graph

The ancestral recombination graph is a fundamental mathematical object, introduced by Griffith and Marjoram [21], modeling all historical genetic exchange events (recombination and coalescent events) that occurs among individuals in the evolving population together with the polymorphisms of the duplication model. Algorithmic approaches to estimate the ARG are discussed in [22].

Given a population of size $N$, the ARG is a directed acyclic graph (DAG) that captures the common evolutionary history of $m$ extant samples [21]. More precisely, an ARG is a random object usually parameterized by four essential parameters: $m$ the number of extant samples, $N$ the population size, $r$ the recombination rate and $g$ the segment length (chromosome or portion of chromosome) carried by each individual. The leaves of an ARG are the extant samples, while the internal nodes are some common ancestors.

The edges represent the flow of genetic materials among individuals from one generation to the next one. The direction is toward the more recent generation (or leaves). In other words, the leaves (extant nodes) have no outgoing edges and the root has no incoming edges. A directed edge from node $v_1$ to node $v_2$ is interpreted as $v_1$ being an ascendant of $v_2$ or $v_2$ is a descendant of $v_1$. The topology has no cycles, since a member cannot be an ancestor of itself. Thus, the topology is always a directed acyclic graph (DAG).

The edges and nodes of the ARG must be annotated with the genetic material that are carrying. The GMRCA plays an important role in restricting the zone of interest in the common evolutionary history of a population. In fact, the GMRCA is defined as an ancestral unit whose genetic material is ancestral to all genetic materials in all the $m$ extant samples.

Fig. 2.3(c) shows an example of ARG. In 2.3(a) is shown the pedigree graph, or complete genealogical network, of a population of size 8. However, if we are interested only in the evolutionary history of 4 extant samples, we can select those samples and trace back their flow of genetic material until we find a root, the GMRCA of the sample (see Fig 2.3(b)). Removing all extraneous parts we obtain a substructure 2.3(c) that is the ARG describing the evolutionary history of 4 samples.

## 2.8 Backward simulators

In the following we report some backward simulators, that have been presented in literature, and we motivate our choice to investigate the development of backward simulation algorithms to estimate the evolution of populations by random graphs.

COSI [23] is an implementation of simulation with the addition of human population demographics to the coalescent model. SelSim [24] is a simulator that incorporates natural selection and recombination within a coalescent framework. SIMCOAL2 [25] allows to simulate the genomic diversity of samples drawn from a set of populations with arbitrary patterns of migrations and complex demographic histories, including bottlenecks and various modes of demographic expansion. Hudson has introduced ms [26] to generate samples drawn from a population evolving according to a Wright–Fisher neutral model. The program assumes an infinite-sites model of mutation, and allows recombination, gene conversion, symmetric migration among subpopulations, and a variety of demographic histories. SimCoal2 and ms have been the most widely applyed, probably due to their flexibility and ease of use [9].

However, backward simulators are suited when deviations from the Wright–Fisher model are minor, for instance they are more limited than forward simulators for modeling

natural selection. On the other hand, even though forward simulators are useful to model more complexity (making them more suited to answer questions at a short timescale) they are much slower because they follow each individual in the population generation by generation. On the contrary, the coalescent approach, used in backward simulators, only considers the genealogy of the samples and not each single individual in the whole population. Recall, indeed, that a forward simulation algorithm (moving in time from past to present while sampling the ARG) may have to simulate the complete network in Fig. 2.3 (a) or (b), whereas the network in Fig. 2.3 (c) is adequate for a backward simulation algorithm (moving in time from present to past). Hence, backward simulations are usually much faster than forward simulations due to the elimination of genetic transmission paths that are not relevant to the samples under study. Backward simulators are generally more efficient specially when population size $N$ is large relative to sample size $m$ [9].

For this reason, we choose to focus on the development of an efficient and accurate backward simulation algorithm that allows to model complex scenarios of multiple population evolution [1] (introduced in Chapter 3). Our algorithm is based on the basic, simplest and most applied backward simulation algorithm, that we call Hudson algorithm or simply Hudson. The latter algorithm is described in the next section. However [10] and [26] give a comprehensive description of Hudson.

## 2.9 Hudson Algorithm

Assume the history of $m$ sampled sequences is being described going backwards in time and that the first event encountered is a recombination event. Before the recombination event, there were $m$ lineages, each carrying the ancestral material to the $m$ samples. Going back in time, after the recombination event, one of the sequences had two ancestral sequences: one carrying ancestral material to the left of the recombination break point and one carrying ancestral material to the right of the recombination point.

If time is measured discretely in generation, the time until a recombination event occurs is geometrically distributed with parameter $r = \rho/N$. Let $T_r$ denote the number of generations until the first recombination event occurs. Tracing a sequence back in time, the probability that this sequence was created by recombination $j$ generations back in the past is:

$$P(T_r = j) = r(1-r)^{j-1}. \tag{2.12}$$

The discrete generations can be approximated with continuous time by means of the exponential distribution. When a recombination event occurs, it is equally likely to occur

in any of the $k$ ancestors and the position of the recombination breakpoint is picked uniformly over the length of the selected sequence.

More precisely, by rescaling time as $t = j/N$, it follows that:

$$P(T_r \leqslant t) = 1 - (1-r)^j = 1 - \left(1 - \frac{Nr}{N}\right)^{Nt} \approx 1 - e^{rNt} = 1 - e^{\rho t}. \tag{2.13}$$

Assuming $k$ sequences, the time to a coalescence event is exponentially distributed with parameter $k(k-1)/2 = \binom{k}{2}$ and the time to a recombination event is exponentially distributed with parameter $k\rho$. Note that these two distributions are independent. Indeed if $N$ is large, it is unlikely that a sequence is involved in both a recombination event and a coalescence event at the same time. Since the two exponential distributions are independent, by using Property 1 of the exponential distribution, the time to the closest event in the past is exponentially distributed with parameter:

$$\binom{k}{2} + \rho k. \tag{2.14}$$

Hence, the probability of a coalescent event is

$$\frac{\binom{k}{2}}{\binom{k}{2} + \rho k}, \tag{2.15}$$

while the probability of a recombination event is

$$\frac{\rho}{\binom{k}{2} + \rho k}, \tag{2.16}$$

where $\rho = rN$.

The algorithm that simulates the process first finds the time until either a coalescence or recombination event occurs. Then, if recombination occurs, the number of ancestral sequences or active lineages $k$ is increased by one, while if coalescence occurs the number of active lineages is decreased by one.

In the following, we describe the main steps of Hudson algorithm for simulating an ARG of a sample of $m$ genomic sequences from the coalescent model with recombination.

**Hudson**

1. Start with $k = m$ samples and active lineages outgoing from the samples;

2. For $k$ active lineages, pick the time to the next event by drawing a random number from the exponential distribution with parameter $\binom{k}{2} + k\rho$. This is the time to the next event;

3. With probability $\frac{\binom{k}{2}}{\binom{k}{2}+\rho k}$ the event is a coalescence event, otherwise it is a recombination event;

4. If it is recombination, pick a random lineages among the $k$ active lineages. Then pick a random point on the sequence carried by the chosen lineage. Create an ancestor lineage with the ancestral material to the left of the chosen point and a second ancestor lineage with the ancestral material to the right of the recombination point. Increase the number of ancestral lineages $k$ by one and go to 1;

5. If it is a coalescence event, choose two lineages at random and merge the sequences into one ancestral sequence and create an ancestral lineage carrying this sequence. Decrease the number of lineages $k$ by one. If $k = 1$ end the process, otherwise go to 1.

The graph structure, resulting from applying Hudson algorithm, is an ARG that includes all information about the history of the sample [10].

In the next chapter we introduce a new backward simulation algorithm, developed in [1], that is based and, therefore, similar to Hudson algorithm described above.

# Chapter 3

# SimRA: Sampling ARG of multiple populations

In this chapter we present the results achieved while investigating randomized methods for studying the evolution of populations. More precisely, we present a backward simulation algorithm which simulates generic scenarios of multiple population evolution with admixture. In the following sections we firstly motivate the study and summarize the main results achieved, then we describe our algorithm SimRA (Simulation based on Random graph Algorithms), that is based on random graphs. Moreover, through comparison studies, we show that SimRA improves dramatically, in time and space requirements, the classical backward simulation algorithm for single population (Hudson algorithm), and produces ARGs in compact forms without compromising any accuracy. Using the underlying random graphs model, we also derive closed form functions of expected values of the ARG characteristics, useful in aiding the user to specify meaningful parameters for complex scenario simulations.

## 3.1 Motivation

Simulating complex evolution scenarios of multiple populations is an important task for answering many basic questions relating to population genomics. Apart from the population samples, the underlying ancestral recombination graph is an additional important vehicle in hypothesis checking and reconstruction studies (see Section 2.7 for a detailed definition of ARG).

In [1] we address the task of modeling and simulating complex scenarios of related multiple populations with subdivision and admixture. These scenarios can be used

to study the effect on the genetic profiles of extant populations as well as for testing complex hypotheses. The aim of simulations is not only to capture the resulting extant populations but also their relevant evolutionary history (for possible reconstruction studies). In literature, most admixture models are based on rather simplistic hypothesis of their possible inter-evolution history. Moreover, one of the main bottlenecks has been the sheer size of the monolithic common history of multi-populations, each of realistic size. Under these conditions simulators of even simple scenarios of only three populations often do not terminate in reasonable time in spite of meaningful parameter settings (sometimes up to 10-12 hours, for instance with COSI [23]). We observed a similar abort-and-rerun requirement in our experiments even with the classical backward simulation algorithm, called Hudson's algorithm (see Section 2.9 for a detailed description of the algorithm).

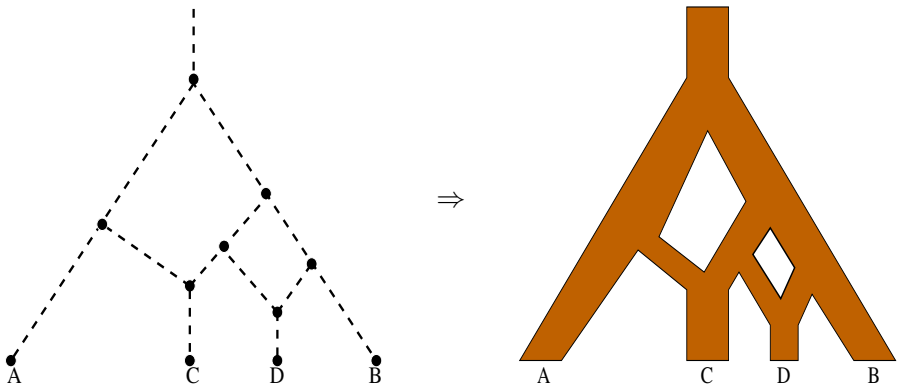
In [1] we present a framework for modeling complex evolutionary scenarios and an algorithm, named SimRA (Simulation based on Random-graph Algorithms), that is both time and space efficient enough to be practical. SimRA makes it possible to run hundreds of experiments in very short time (in minutes) enabling a very effective means of carrying out complex studies, such as in [3].

SimRA is based on random graph and backward simulation of the ARG. Recall that backward simulations begin in the present and move in time through the past generations and are usually more efficient than forward simulations due to the elimination of many (obvious) redundant paths in the evolution process. See Sections 2.1 and 2.8 for a detailed comparison between backward and forward simulation algorithms. The big picture showing the relationship between a complete genealogical network (or pedigree graph) and an ARG highlighting the backward trace of evolutionary history of a sample is illustrated in Fig 2.3.

## 3.2 Modeling multiple population evolution

We model the relationship between $m$ populations by a Direct Acyclic Graph (DAG) $P'$ with $m$ leaf nodes, and we call it a *scaffold*. An example is shown in Fig 3.1 (i). The progress of time is assumed to be from top to bottom and the $m$ leaf nodes are annotated with the population labels. Further, each edge $e$ in $P'$ has three characteristics:

- $\text{len}(e)$ : the incubation length, that is a time parameter defined in generations;
- $l_b(e)$ : the number of the lineages at the bottom of the edge;
- $l_t(e)$ : the number of lineages at the top of the edge.




(i) scaffold $P'$ (ii) ARG $P$

FIGURE 3.1: An example with four populations A, B, C, D. (i) shows the scaffold $P'$. (ii) shows a corresponding ARG $P$. Note that in general the structure of $P'$ is not apparent from $P$ and the ARG $P$ simply looks like the ARG in Fig 2.3. See text for more details.

Note that two parameters, an effective population size $N$ and a recombination rate $r$, determine the number of lineages $l_t(e)$ for a fixed pair of values of $l_b(e)$ and $\text{len}(e)$.

We assume that the scaffold $P'$ is binary, meaning that each internal node in $P'$ has exactly two ascendants or two descendants, but not both.

For each internal node, we define two *junction constraints* as follows. A node $v$ in $P'$ is called *split* node if it has two incoming edges $e_1$ and $e_2$ and an outgoing edge $e_3$. For a split node $v$, the following relationship holds: $l_t(e_3) \leqslant l_b(e_1) + l_b(e_2)$, i.e., the lineages at $v$ is the union of the lineages of the two incoming edges. Similarly a node $v$ is *merge* node if it has two outgoing edges $e_1$ and $e_2$ and one incoming edge $e_3$, then $l_b(e_3) \leqslant l_t(e_1) + l_t(e_2)$ holds, i.e., the lineages at $v$ is the union of the lineages of the two outgoing edges.

Each edge $e$ of $P'$ represents the evolution of a Wright Fisher population captured in a DAG say $P_e$. The union of each of these DAGs by appropriately gluing the ends of the edges corresponding to the nodes of $P'$ gives the ARG $P$ that can be written as: $P = \bigcup_{e \in P'} P_e$. Such a $P$ is shown in Fig 3.1 (ii) where the leaf nodes correspond to extant units of each population of $P'$.

Finally, we say that $P'$ defines *admixture* if there exists a closed path (CP) in $P'$.

Fig 3.2 shows an example of parameters that define the scaffold $P'$. Additionally a recombination rate ($r$) and effective population size for each edge, ultimately decides the topology of the resulting ARG. Further, the mutation rates and the short tandem repeats (STRs) details define the polymorphism in the samples of the individuals of the populations.

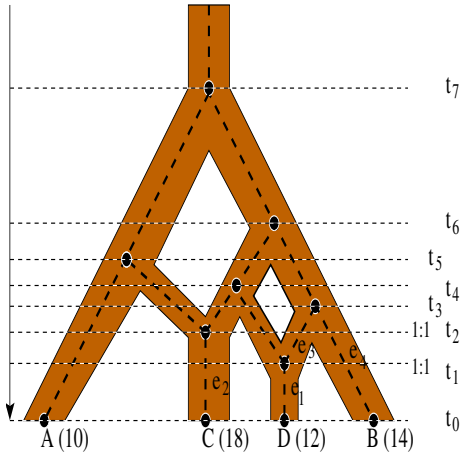



FIGURE 3.2: Specifying the family of 4 populations, A, B, C and D with sample sizes 10, 14, 18 and 12 respectively. The horizontal dashed lines correspond to times $t_0 = 0 < t_1 < t_2 < ... < t_7$. At times $t_1$ and $t_2$ the surviving lineages are split in the ratio 1:1 along the diverging lines of the scaffold at the split nodes.

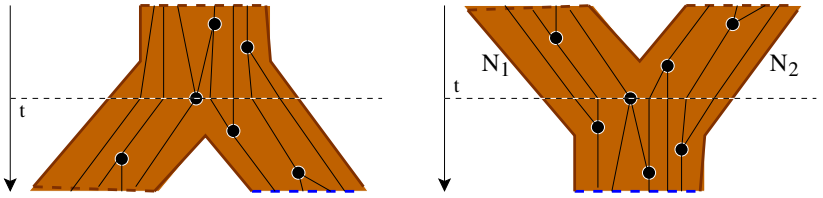
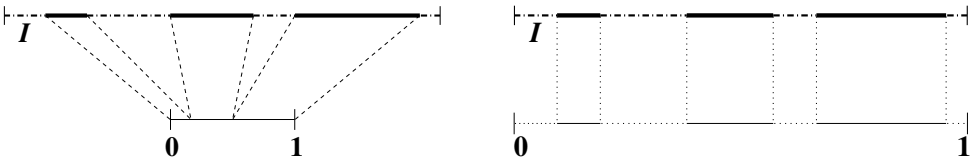



FIGURE 3.3: The two types of nodes in a scaffold $P'$ with thick brown edges: Merge and split nodes are marked by the horizontal dashed lines at time $t$. The tiny black disc nodes and the thin black edges are part of the underlying ARG $P$.

Consider the scaffold specified in Fig 3.2. Each edge is simulated as a single population. In Section 3.3 we describe our algorithm for sampling an ARG of a single population. Assume that the effective populations size $N_{e_j}$ is specified for each edge $e_j$. For instance, the edge $e_1$ labeled with population $D$ is simulated with 12 extant samples, i.e., $l_b(e_1) = 12$, $len(e_1) = t_1$. The resulting surviving lineages $l_t(e_1)$ is split in the ratio 1:1 as shown in Fig 3.3. Similarly, the edge $e_2$ is labeled with population $C$ with 18 extant samples, $l_b(e_2) = 18$, $len(e_2) = t_2$. The resulting surviving lineages $l_t(e_2)$ is split in the ratio 1:1. Next, $l_t(e_1)/2$ lineages are simulated until a time depth of $t_3$ on edge $e_3$ to give $l_t(e_3)$ lineages. Population $B$ is simulated with 14 extant samples on edge $e_4$ until a time depth of $t_3$, i.e., $l_b(e_4) = 14$, $len(e_4) = t_4$. The $l_t(e_3)$ lineages are combined with $l_t(e_4)$ lineages. This node in the scaffold is a merge node as shown in Fig 3.3 and the population is simulated to a time depth of $t_6$ (i.e., for time $t_6 - t_3$). Similarly all the edges are simulated until a total time depth of $t_7$.




(a) Using den($I$) in Eqn 3.16. (b) Using len($I$) in Eqn 3.16.

FIGURE 3.4: The top line represents a chromosomal segment $I$ carried by an edge in both (a) and (b): gaps($I$) are shown as dashed lines and solid($I$) as solid segments. $I$ is mapped to a normalized line segment, say $[0, 1]$, shown in the bottom in both (a) and (b). In (a) the gaps are skipped, and the lengths of each element in solid($I$) is proportionally represented in $[0, 1]$. Thus any element in $[0, 1]$ can be mapped back to a unique location in solid($I$). In (b) the gaps are not skipped, and the lengths of each element in solid($I$) and in gaps($I$), is proportionally represented in $[0, 1]$. Any element only in the solid section in $[0, 1]$ can be mapped back to a unique location in solid($I$); any other element maps to a gap in $I$.

## 3.3 Modeling single population evolution

Now we address the problem of sampling an ARG of a single population. This is a well studied problem as discussed in [10, 26]. The core of SimRA, that simulates the evolution of a single population, is based on Kingman coalescence [18] and Hudson algorithm (that we described in Section 2.9). The latter algorithm is not efficient enough to admit complex multiple population simulations: it is too time consuming and in many instances failed to terminate in reasonable time, forcing to abort and re-run. Simulating the evolution of multiple populations requires multiple runs (corresponding to each population, i.e., each edge in a scaffold) thus making the Hudson prohibitively expensive.

In the following, we present SimRA for simulating a single (neutral) population that makes a few subtle changes in the algorithm based on Observation 1 (as we describe later in Section 3.4).

The algorithm works back-in-time starting from the present (time 0), moving back into the past. Further, the ARG is incrementally constructed by identifying the event nodes (recombination or coalescence) in the graph. An event node either has multiple incoming or multiple outgoing edges. For example a chain node is not an event node. An important assumption, that considerably simplifies the algorithm, is made: the probability of multiple events in the same epoch (generation) is extremely low, hence the algorithm assumes there is at most one event per generation. The design of the overall algorithm is affected by this and at each step the algorithm simply seeks the closest generation from the current one where an event node occurs.

In the remained of the chapter we say that a lineage is *active* if it has no node father and it can be involved in a new coalescent o recombination event.

To keep the discussion self-contained, we recall here a few basic definitions that has been presented in Chapter 2 in more details.

### Basic definitions

**Coalescent event.** If the population size is $N$ then the probability of two units having the same ancestor in the immediate previous generation is $1/N$. Thus the probability of having the same ancestor exactly at $j$ generation ago is $\left(1-\frac{1}{N}\right)^{j-1}\frac{1}{N}$. In other words, the probability of two units having same ancestor at generation $> j$ ago is

$$P_{J\geqslant j} = \left(1-\frac{1}{N}\right)^{j}. \tag{3.1}$$

Notice that there is no biology at play here. Simply the stochastic nature of the transmission of genetic material through generations in a population. Using

$$j = Nt \tag{3.2}$$

to convert the equation to the form of Eqn 2.10, we obtain the corresponding exponential distribution ($\lambda = 1$) as follows:

$$\left[P_{T\leqslant t} = 1-\left(1-\frac{1}{N}\right)^{Nt}\right] \approx \left[P_{T\leqslant t} = 1-e^{-t}\right]. \tag{3.3}$$

**Recombination event.** Let a recombination event occur uniformly on a chromosomal segment. Note that each unit carries a genetic material of some length. Let $r_l$ be the probability of a recombination in one generation of lineage $l$ (depending on the segment in the chromosome this unit carries). The probability of this unit having encountered its first recombination event exactly $j$ generations ago is

$$(1-r_l)^{j-1}r_l.$$

In other words, the probability of the unit having a recombination at generation $> j$ ago is

$$P_{J\geqslant j} = (1-r_l)^{j}. \tag{3.4}$$

Indeed there is biology, i.e., recombination, at play here. Thus both the biology and the stochastic processes will affect the shape of the ARG.

Letting $j = Nt$, so that the scaling matches the time scaling for the coalescence event, and using Eqn 2.10 (since $r_l$ is small and $N$ is large), we have:

$$\left[P_{T\leqslant t} = 1 - \left(1 - \frac{Nr_l}{N}\right)^{Nt}\right] \approx \left[P_{T\leqslant t} = 1 - e^{-(Nr_l)t}\right]. \tag{3.5}$$

**Approximation to exponential distributions.** The approximations to exponential distribution in Eqns 3.3 and 3.5 help in designing a time-efficient (approximate) simulation algorithm. But these approximation can be used only when $N$ is large and $r_l$ is small.

**Genetic material flowing through the ARG.** Based on the following Observation 1 from [19], we describe how the nodes and the edges in the ARG have to be decorated with the genetic material and how the latter is transmitted through the ARG.

**Observation 1.** *(***ancestor without ancestry paradox***) The edges (and nodes) of an ARG must be annotated with the chromosomal segment that flows through the edges.*

The chromosomal segment whose evolutionary history is captured by the ARG is represented as the real interval $[0, 1]$, without loss of generality. The implicit assumption is that this maps to the chromosome of length $g$. Every node in the ARG is annotated with some sub-interval or union of sub-intervals of $[0, 1]$.

Thus the genetic material, $I$, carried by a node is: $I = \{[\ell_1, u_1], [\ell_2, u_2], ..., [\ell_s, u_s]\}$, where $0 \leqslant \ell_1 < u_1 < \ell_2 < u_2 < .. < \ell_s < u_s \leqslant 1$. The closed intervals $[\ell_i, u_i] \in I$ are termed *solids* and the open intervals $(u_i, \ell_{i+1})$ are termed *gaps* where $1 \leqslant i < i + 1 \leqslant s$. The length (len) of $I$ is defined as the total span of $I$, irrespective of the gaps, while the density (den) of $I$ is defined as the total span of the solid intervals only. The definitions are summarized as:

$$\begin{aligned}
\text{solid}(I) &= [\ell_1, u_1] \cup [\ell_2, u_2] \cup ... \cup [\ell_s, u_s], \\
\text{gaps}(I) &= (u_1, \ell_2) \cup (u_2, \ell_3) \cup ... \cup (u_{s-1}, \ell_s), \\
\text{len}(I) &= u_s - \ell_1, \\
\text{den}(I) &= \sum_{i=1}^{s} u_i - \ell_i, \\
[x, y] \subset \text{solid}(I) &\Leftrightarrow [x, y] \subset [\ell_i, u_i], \text{ for some } 1 \leqslant i \leqslant s.
\end{aligned}$$

The union, or merge, operation on segments, $I_a \cup I_b = I_{a\cup b}$, has the natural interpretation:

$$[\ell, u] \in I_{a\cup b} \Longleftrightarrow [\ell, u] \subset \text{solid}(I_a) \text{ OR } [\ell, u] \subset \text{solid}(I_b). \tag{3.6}$$

The splitting of a segment $I$ at point $x$ ($\ell_1 \leqslant x \leqslant u_s$) into $I_a$ and $I_b$ is defined as:

$$I \xrightarrow{\text{split}} \begin{cases} I_a = I_b = I, \text{when } x = \ell_1 \text{ or } x = u_s, \\ \left.\begin{aligned} I_a &= \{[\ell_1, u_1], [\ell_2, u_2], ..., [\ell_j, x]\}, \\ I_b &= \{[x, u_j], [\ell_{j+1}, u_{j+1}], ..., [\ell_s, u_s]\}, \end{aligned}\right\} \text{when } \ell_j < x < u_j, \\ \left.\begin{aligned} I_a &= \{[\ell_1, u_1], [\ell_2, u_2], ..., [\ell_j, u_j]\}, \\ I_b &= \{[\ell_{j+1}, u_{j+1}], ..., [\ell_s, u_s]\} \end{aligned}\right\} \text{when } u_j \leqslant x \leqslant \ell_{j+1}. \end{cases} \tag{3.7}$$

Note that if a coalescent event occurs involving two lineages $l_a$ and $l_b$, then their respective genetic materials $I_a$ and $I_b$ are merged as described by Eqn 3.6. The resulting genetic material $I_{a\cup b}$ will be carried by the new active lineage $l$ created by the coalescent event.

On the other hand, if a recombination event occurs at lineage $l$ the genetic material $I$ carried by $l$ is split as described by Eqn. 3.7. The new two active lineages generated by the recombination event $l_a$ and $l_b$ will carry respectively two segments $I_a$ and $I_b$ resulting from the splitting.

In the next paragraph we describe how to compute the time of the closest event in the past and decide the kind of this event.

**The closest event node from the current state in the ARG.**

Let $L$ lineages be active at time $T$. Since these lineages are active they can be involved in a coalescent or recombination event. Let $t_{ab}^{\text{coal}}$ denote the time to the coalescence of lineages $l_a$ and $l_b$. Let $t_l^{\text{rcmb}}$ denote the time to the closest (to $T$) recombination event of lineage $l$. Eqn 3.1 shows that each of the $\binom{L}{2}$ coalescent events, generically written as $t_{ab}^{\text{coal}}$, can be approximated by an exponential distribution with parameter $\lambda = 1$.

On the other hand, based on the Observation 1, Eqn 3.5 can be approximated by an exponential distribution with parameter $r'_l$ where $r'_l = Nr_l$ and $r_l$ is the recombination rate of the segment flowing through lineage $l$. Hence, each active lineage $l$ has is own recombination rate $r_l$ that is given by $r_l = g\text{len}(I)$, where $I$ is the genetic material carried by $l$ and $r$ is the recombination rate (i.e., input parameter of the algorithm).

These approximations to the exponential distributions are based on two assumptions of the Wright Fisher population: the population at each generation is $N$ and a unit picks its parent randomly from the previous generation (non-overlapping generations and panmictic mating population). Also, note that the factor of $N$ in $r'_l$ is due to the approximation of the distributions, and not due to the underlying population evolution model.

The task is to find $t$, the time to the closest event node in the past. This event could either be a coalescent event (merging of two lineages) or a recombination event (splitting of a lineage in two new lineages). Since all the events are independent, then we seek overall minimum. Thus using Property 1 of exponential distribution, we compute the time $t$ to the closest event back in time from the current time $T$ as following:

$$\begin{aligned} t &= \min\left(\overbrace{\min_{1\leqslant a<b\leqslant L}(t_{ab}^{\text{coal}})}, \underbrace{\min_{1\leqslant l\leqslant L}(t_l^{\text{rcmb}})}\right) \\ &= \text{Exp}\left(\overbrace{1+1+\cdots+1}+\underbrace{r'_1+r'_2+\cdots+r'_L}\right), \end{aligned} \tag{3.8}$$

The over-braces capture the $\binom{L}{2}$ possible coalescent events and the under-braces capture the $L$ possible recombination events. In order to understand if the closest event is coalescence or recombination we pick the minimum value using Property 2 of exponential distribution.

Hence, the event is a coalescent event with probability

$$\frac{\binom{L}{2}}{\binom{L}{2}+\sum_l r'_l}, \tag{3.9}$$

and a recombination event at lineage $1 \leqslant k \leqslant L$ with probability

$$\frac{r'_k}{\binom{L}{2}+\sum_l r'_l}. \tag{3.10}$$

### 3.3.1 Algorithm to generate the topology

`INPUT`: The input parameters and some typical parameter values for a human chromosomal segment are given in Table 3.1. Note that due to historical reasons, the unit of recombination rate is specified in centiMorgans per megabase per generation and the mutation rate is specified in number of mutations per base pair per generation ($\times 10^{-8}$). Recall that the relationship between generation $j$, (discrete) and the time $t$ (or edge length), used in the Kingman coalescence, is defined as $j{=}Nt$. See Section 2.5 for more details.

`ASSUMPTION`: Not more than one event, coalescent or recombination, occurs at a generation. Also, no back mutations, i.e., a position (base) undergoes no more than one mutation in the entire ARG. The mutation rate and recombination rate are uniform over the segment

| | Parameters | User-Specified Units | Units in bp for Algorithm | Example Values |
|---|---|---|---|---|
| $g$ | segment length | Kb | $\times 10^3$bp | $75 \times 10^3$ |
| $V$ | STR locations | - | - | $[0.3, 0.7] \times g$ |
| $m$ | extant units | - | - | $100 \times 1$ |
| $N$ | population size | - | - | $10 \times 10^3$ |
| | rates/generation | | | |
| $r$ | recombination | cM/Mb/gen | $f(x) = \begin{cases} \times(\frac{0.01}{10^6} = 10^{-8}) \\ \text{Morgan/bp/gen} \end{cases}$ | $0.1 \times 10^{-8}$ |
| $\mu$ | SNP mutation | mut/bp/gen $\times 10^{-8}$ | $\times 1$ mut/bp/gen | $1.5 \times 10^{-8}$ |
| $\mu^{str}$ | STR mutation | mut/locus/gen $\times 10^{-4}$ | $\times 1$ mut/locus/gen | $6.9 \times 10^{-4}$ |

TABLE 3.1: The table shows some typical parameter values for a human chromosomal segment that SimRA takes as input.

being simulated.

`OUTPUT`: ARG; $L$ is the number of GMRCAs.

`ALGORITHM`:

**Initialization.**

1. The genetic material, $I_v$, of each of the $m$ leaf nodes, $v$, is set to $I_v = \{[0, 1]\}$. The number of active lineages $L$ is initialized to $m$.

2. For lineage $l$, incident on leaf node $v$, the recombination rate (using Eqn (3.5))

$$r'_l = Nr_l \text{ where } r_l = gr\text{len}(I_v). \tag{3.11}$$

   Since, $\text{len}(I_v) = 1$ for the leaf nodes, for each $l$, $r'_l = \alpha$ where

$$\alpha = Ngr. \tag{3.12}$$

3. Time $T$ is set to 0 and iteration $i$ to 1.

**Loop.** Iterate until $L$ is one (or $T$ crosses a pre-defined threshold).
Iteration $i$ is defined as follows.

1. Compute the recombination rate $r'_l$ of each lineage $l$ (the outgoing edge on node $v$) using Eqns 3.11 and 3.12 as $r'_l = \alpha \times \text{len}(I_v)$. Then compute the time $t_i$ to the

next event using the exponential distribution (Eqn 3.8):

$$t_i = \text{Exp}\left(\binom{L}{2} + \sum_{l=1}^{L} r'_l\right). \tag{3.13}$$

In other words, draw a random number from the above exponential distribution.

2. Based on Eqns 3.9 and 3.10, if coalescent event, then pick two lineages, $l_a$ and $l_b$ (with genetic material $I_a$ and $I_b$ respectively) at random and coalesce them to one and update the genetic material of this new node and lineage $I_a \cup I_b$ (as defined in Eqn 3.6). Update $L$ to $L-1$.
   If recombination at lineage $l_k$, then randomly pick a point $x$ on the segment being carried by lineage $k$, splitting the lineage into two, as defined in Eqn 3.7. Update the genetic material of the two lineages based on this splitting point. Update $L$ to $L+1$.

3. $T$ is updated as $T + t_i$ and iteration as $i + 1$.

### 3.3.2 Painting ARG edges with SNP & STR mutations

We consider two kinds of polymorphisms that can occur in the genomic sequences: Single Nucleotide Polymorphisms (SNP) and Short Tandem Repeats (STR) mutations.

Note that the mutations (SNPs and STRs) do not affect the shape of the neutral ARG (i.e., ARG sampled without considering selection). Thus after the ARG has been fully constructed, the edges can be annotated with the mutations.

#### SNP Mutations

SNPs are variations in DNA sequences occurring commonly within a population of individuals in which a single nucleotide — A, T, C or G — in the genome (or other shared sequence) differs between members of a biological species or paired chromosomes. For example, two sequenced DNA fragments from different individuals, AAGCCTA to AAGCTTA, contain a difference in a single nucleotide.

For SNP mutations, we assume the infinite sites model, meaning that no more than one mutation event occurs at each site. The mutation rate, $\mu$, is specified in terms of generation per site, and, the assumption is that the rate holds uniformly along the chromosome. Thus the probability $p$ of mutation at one site is the modified mutation rate for $j$ generations as

$$p = \mu j. \tag{3.14}$$

Consider a segment $I_v$ carried by an edge incident on node $v$. Note that den($I_v$) denotes the span of the solid intervals in this segment (without gaps). If $j$ is the number of generations represented by this edge, then the expected number of mutations, $\lambda$, in this segment after $j$ generations, is

$$\lambda = p \text{ den}(I_v) = \mu j \text{ den}(I_v). \tag{3.15}$$

Usually since $p$ is small, den($I_v$) is large and the product is a small number, the distribution of this number can be approximated by a Poisson distribution. Hence in the algorithm, we use a Poisson distribution with parameter $\lambda$ to estimate the number of mutations that occur after $j$ generations and decorate the segment carried by the edge incoming on node $v$.

Let time $t$ be associated with an incoming edge on node $v$. At this stage, each edge is annotated with the mutation events, which is appropriately reflected in $I_v$, the segment carried by node $v$. Since number of generations is $Nt$ and the span of the segment $I_v$ has been normalized in the initialization step, let

$$p = \mu N t \text{ and } n = g\text{den}(I_v). \tag{3.16}$$

Each edge of the ARG, incident on node $v$, is annotated with number of mutations based on Eqn 3.16 as follows. $X$, the random draw from a Poisson distribution with parameter $np$:

$$X = \text{Poisson } (np). \tag{3.17}$$

Afterwards, the actual locations of the $X$ mutations are placed at random in the solid intervals of $I_v$ carried by the edge incoming on node $v$ (excluding the gaps, see Fig 3.4).

### STR mutations

STRs are short sequences of DNA, normally of length 2-5 base pairs, that are repeated numerous times (typically 5-50 times) in a head-tail manner. For example, the 16 bp sequence of GATAGATAGATAGATA would represent 4 head-tail copies of the GATA. The polymorphisms in STRs are due to the different number of copies of the repeat element that can occur in a population of individuals. STRs occur at thousands of locations in the human genome and they are notable for their high mutation rate and high diversity in the population.

Note that the number and positions of the STR loci are fixed by the input specification. For each STR locus, $k$, carried by the segment $I_v$, we compute the following. The number

of STR mutations at locus $k$ on each edge of the ARG, incoming on node $v$, is $X_v$, the random number draw from a Poisson distribution with parameter $Nt\mu^{\text{str}}$:

$$X_k = \text{Poisson } (Nt\mu^{\text{str}}). \tag{3.18}$$

Note that $\mu^{\text{str}}$ is the mutation rate for STRs given in input. Let $p_+$ be the probability of the mutation that increases the number of copies (by 1 in one generation) and $p_-$ be the probability of the mutation that decreases the number of copies (by 1 in one generation). Then, $X_{k_+}$, the number of times the STR mutation results in an increase in the number of copies of the repeat follows a binomial distribution, hence is the random draw from a binomial distribution with parameter $X_k$ and $p_+$ $X_{k_+} = \text{Binomial } (X_k, p_+)$. Thus the remainder, i.e., $X_k - X_{k_+}$ must be the number of events that result in decrease of the number of copies. Thus $\Delta_k$ the net change in the number of copies at locus $k$ is: $\Delta_k = X_{k_+} - (X_k - X_{k_+}) = 2X_{k_+} - X_k$. If unspecified, we use the default value of $p_+ = \frac{1}{2}$, assuming $p_+ = p_- = \frac{1}{2}$.

## 3.4 Difference from Hudson algorithm

In an implementation of SimRA algorithm, both Eqns 3.9 and 3.10 are used in a single draw of a random number. Imagine a unit interval $[0, 1]$ is broken up into $1 + L$ sub intervals of lengths in the following ratio $\binom{L}{2} : r'_1 : r'_2 : ... : r'_l : ... : r'_L$. Thus a random number drawn from the interval $[0, 1]$ belongs to one of these $1 + L$ sub-intervals and is appropriately interpreted. In other words, the first interval implies coalescent event and $k$th ($k > 1$) interval implies a recombination at the lineage $l_{k-1}$.

Note that Steps 1 and 2 in SimRA differ from the corresponding steps in Hudson algorithm Indeed in Hudson algorithm, a single scaled recombination rate $\rho = Nr$ is used throughout the iterations in the backward simulation of the ARG, , while SimRA uses the L segmented versions $r'_l = Nr_l$, for $l = 1, \ldots, L$. This is reflected in Equation (3.9) in SimRA that differs from Equation (2.15) in Hudson and in Equation (3.10) in SimRA that differs from Equation (2.16) in Hudson.

In our algorithm Equation (3.8) suggests that to account for recombination event, the time of the closest event $t$ takes into account not just the number $L$ of active lineages but also the length of the segments carried by each of them (i.e., $r_l = grlen(I_v)$ of Equation (3.11)).

Note that $\sum_v len(I_v) \neq 1$ at each iteration makes the two computations distinct; hence the algorithms SimRA and Hudson. A consequence is that in Hudson algorithm if $p_c$ is

the probability of coalescence (Equation (2.15) ) then the probability of recombination is $1 - p_c$ with equal probability over all the lineages, meaning that a lineage is picked at random among the active ones. On the other hand, SimRA uses Equation (3.10) to pick the lineage for recombination (See Step 2 of SimRA algorithm). Thus Equation (3.10) has no counterpart in the classical Hudson algorithm. The accuracy of the two algorithms are comparable, while SimRA outperforms Hudson in time, space and non redundancy factor as we will show in Sections 3.6, 3.7.3 and 3.10.

## 3.5 Comparison study

For a comprehensive survey of sampling algorithms and simulators the reader is directed to [9], which discusses both backward and forward simulators that incorporate a variety of demographic models. SimRA simulates multiple populations under admixture and subdivision, while other simulators incorporate other demographic models, making the comparison difficult. However, the core engine of SimRA can be compared with Hudson, which forms the basis in all backward simulators. Hence, in the comparative analysis, we run SimRA for sampling the ARG of a single population version of (the algorithm core described in Section 3.3 and the classical Hudson's algorithm, presented in Section 2.9.

Furthermore, to keep the comparisons agnostic to the underlying systems, we use identical implementation for the common parts of SimRA and Hudson. In Section 3.6 we compare SimRA and Hudson on time and space performance. In Section 3.7.3 we compare the compactness of the ARGs produced by both algorithms. Finally in Section 3.10 we compare their accuracy.

## 3.6 Time and space performance

We performed extensive comparative analysis between SimRA and Hudson algorithms for a single population. In particular, we carried out hundred runs for each parameter set up, for both the algorithms. Fig 3.5 shows the superior performance of SimRA in both time and space. The difference is particularly accentuated with increasing values of recombination rate $r$. Notice that for higher values of $r$, the time and space requirement is nearly two orders of magnitude higher for Hudson, while SimRA time and space requirements are at par or lower than Hudson for $r = 0.1 \times 10^{-8}$. It is worth pointing out that $r = 0.3 \times 10^{-8}$ is the value of recombination rate of real data.

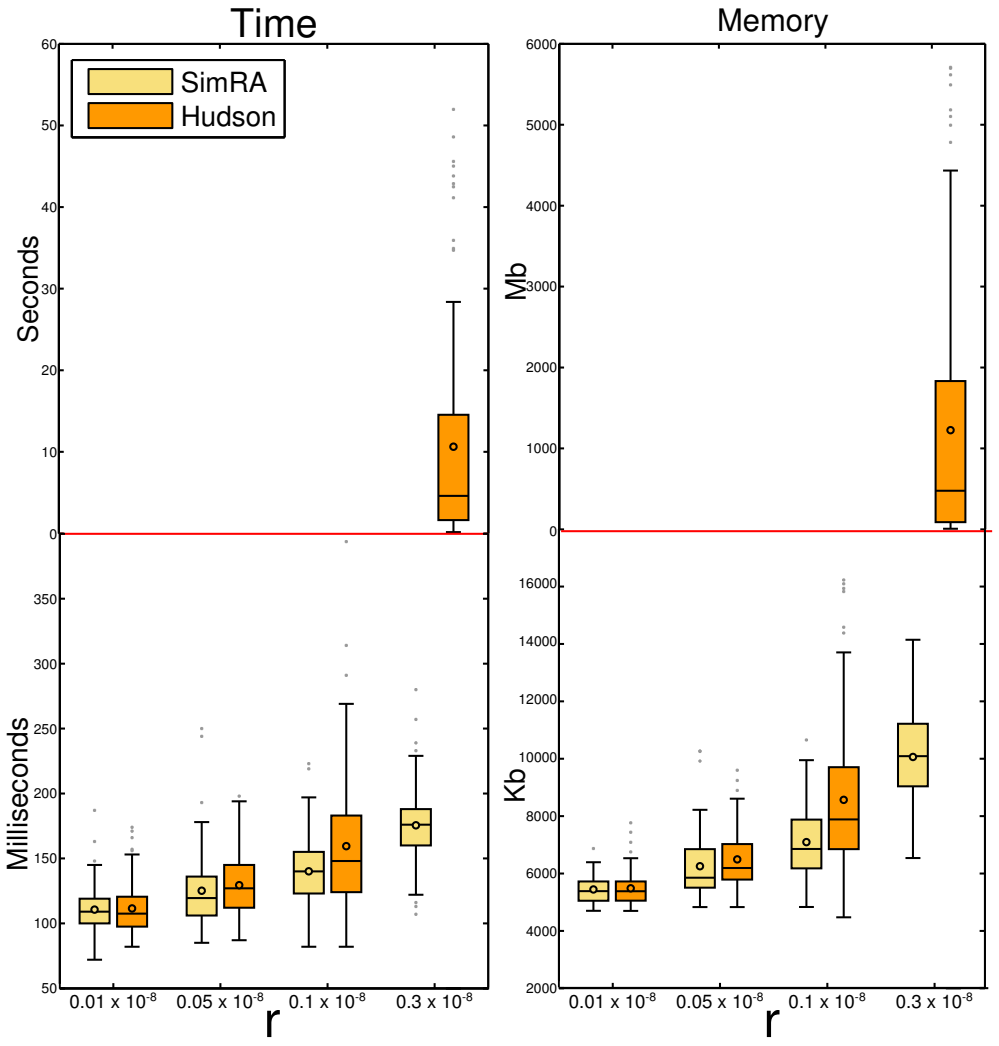



FIGURE 3.5: The box-and-whisker diagrams of the time and memory performance of SimRA and Hudson computed for 100 runs with N=10K, g=150K, $\mu = 1.5 \times 10^{-8}$ and different value of recombination rates (shown in the x-axis). On each box, the central mark is the median, the circle is the mean, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. The red line demarcates the time and space requirement for Hudson for the rightmost value of $r$. While for lower values of $r$ the time and space requirement of SimRA is marginally lower than that requireq for Hudson, notice that for the high value of $r$, the time and space requirement for SimRA is nearly two orders of magnitude lower than that for Hudson.

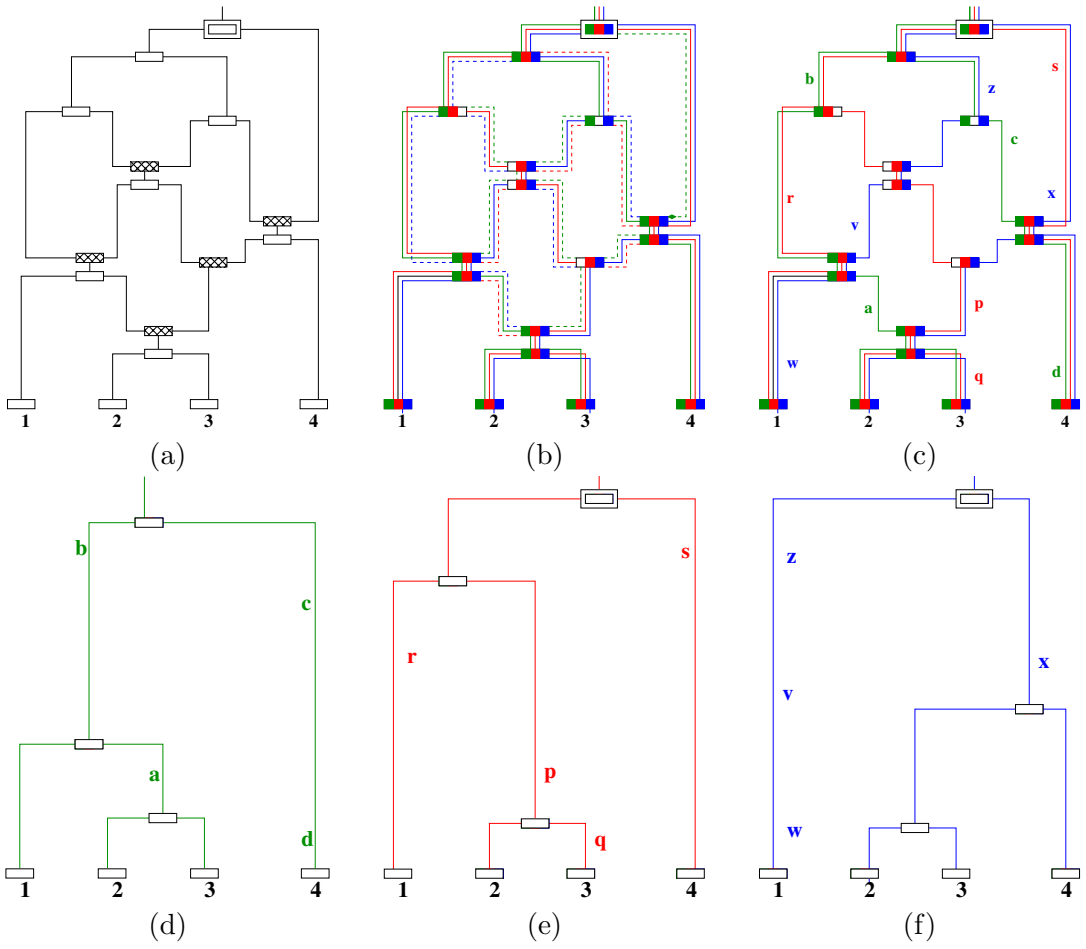



FIGURE 3.6: (a) The ARG of Fig 2.3. (b) shows the flow of the 3 non-mixing segments. These are shown in three distinct colors, green, red and blue (from left to right on the chromosomal segment). The dashed edges imply that these do not affect any of the 4 extant samples, due to a recombination node in their path. (c) The ARG showing only those segments that affect at least one of the 4 extant samples. Note that for each color, the corresponding subgraph is a tree, called marginal or embedded tree. These trees are shown in (d)-(f).

## 3.7 Compactness measure of ARG

In this section we compare the compactness of the ARGs produced by SimRA algorithm with the ARGs produces by Hudson algorithm [10].

In order to carry out this comparison, we define the compactness of an ARG in terms of its non-redundant core, the minimal descriptor introduced in [27]. Hence, we discuss the reduction of a given ARG to a minimal descriptor by removal of non t-coalescent nodes. These are nodes that have a single ascendant and multiple descendants while they are not non-trivial nodes, i.e., part of a chain, in any of the embedded marginal trees. In other words a non t-coalescent node is a coalescent node in the ARG, but it is not a coalescent node in any of the embedded marginal trees (see [27] for a detailed

exposition). In [1] we prove that the minimal descriptor of an ARG $G$ is unique and we use it to compare the compactness of the ARGs produced by SimRA and Hudson.

### 3.7.1 Background

To keep the discussion self-contained, in the following we paraphrase some important definitions (t-coalescent node and minimal descriptor), theorems and lemmas from [27].

Let $G$ be an instance of the random object ARG. Let the edges of $G$ be directed from the root to the more recent generation (or the leaves). The edges of the ARG are annotated with genetic event (mutations) and the labels are displayed in the illustrations. See Fig. 3.6 for an example. The ARG $G$ is defined on $m$ extant samples and is decomposed into $K$ trees, where $K$ corresponds to the number of un-mixing or completely-linked segments in the extant samples. These un-mixing units are portions of the whole segment that are never split by recombination events during the evolution. Note, indeed, that $K$ is equal to the number of recombination events minus one.

**Theorem 3.1.** *Every ARG $G$ on $m > 1$ extant samples is the topological union of some $K \geqslant 1$ trees (or forests) called* marginal *or* embedded *trees.*

An edge in $G$ is defined to have multiple strands that are shown as distinct colors, where each color corresponds to one of the component trees $1 \leqslant i \leqslant K$. Between any pair of vertices $v_1$ and $v_2$ in $G$, no two strands can be of the same color. Thus the number of multiple strands, corresponding the edge, between a pair of vertices $v_1, v_2$ can be no more than the number of marginal trees $K$. Each strand of an edge is labeled by a set of genetic events (SNPs), possibly empty. The annotations on the edges (with SNPs and/or STRs) play a critical role since they ultimately shape the extant samples. To keep this discussion simple, let the non-exchange genetic event correspond to SNPs.

Note that the embedded or marginal trees are very important in an ARG and critical in defining it. Given the topology of an ARG $G$ with the embedded trees and the annotations representing genetic duplication events, an unambiguous genetic flow giving rise to the samples $S(G)$ is defined. The set of labels of edge $v_1v_2$ is written as $lbl(v_1v_2)$. Then $x_i \in lbl(v_1v_2)$ is a label on strand $i$ of edge $v_1v_2$. For example in Fig. 3.6 the labels on the green tree are the SNPs a, b, c, d.

The exact position of the SNPs on the genome does not matter. However, in the ARG, a particular ordering of the $K$ trees is assumed. Hence the SNPs of each of the $K$ trees respect this order. More precisely, we assume that the un-mixing $K$ segments of interest are consecutive on the chromosomes of the samples. Hence, the corresponding embedded

trees can be numbered by consecutive integers from 1 to $K$. An example is shown in Fig 3.6, where the red marginal tree represents the flow of the the leftmost segment and blue one of the rightmost. Observe that this is then reflected in the sample definitions.

The samples of a graph instance $G$ of the ARG are denoted as $S(G)$. The latter is a set of $m$ sequences which is also the number of leaf nodes (or extant units) in $G$. Each sequence is obtained simply by flowing the genetic event labels (mutations) of tree $i$, for each $1 \leqslant i \leqslant K$, along paths of color $i$ all the way down to the leaf units (samples).

The multiple colored edges of $G$ implies an annotation of a node $v$ as well. In [27] the annotation of a node $v$, $sg(v)$ is defined as follows.

**Definition 3.2** ($sg(v)$ overlap)**.** Given node $v$ in an ARG G, $sg(v)$ is the set of the embedded trees that $v$ is incident on. Two vertices $v_1$ and $v_2$ in $G$ are said to overlap if $sg(v_1) \cap sg(v_2) \neq \emptyset$.

In order to identify redundancies in the topology of an ARG $G$, nodes that do not effect the topology on the samples $S(G)$ have been identified. The removal of these nodes in the ARG leads to a core that preserves the essential characteristics.

More precisely the removal of a node $v$ from an instance $G$ of ARG is defined by the following steps [15]:

1. For each child $v_{c,i}$ of $v$, that is in the embedded tree $1 \leqslant i \leqslant M$
    a. *adding new edges:* connect $v_{c,i}$ by a new edge to $v_{p,i}$, a parent of $v$ in the marginal tree $i$;
    b. *annotating the new edges.*: annotate the new edges between $v_{c,i}$ and $v_{p,i}$ as follows: for each strand $i$, the label of the new edge is the union of the labels on the i-path from $v_{p,i}$ to $v_{c,i}$. Next, if a label appears on multiple new outgoing edges of $v_{p,i}$, then it is removed from all but one of the outgoing edges. (This is to avoid introducing mutations, i.e., the same label appearing multiple times on the embedded tree $i$.)
2. The node $v$ with all the edges incident on it are removed from $G$.

**Definition 3.3** (sample preserving)**.** Two distinct instances of ARG, $G$ and $G'$, are samples preserving if and only if $S(G) = S(G')$.

When two ARGs are samples preserving, all the allele statistics, including allele frequencies, linkage disequilibrium (LD) decay, and so on are identical in the two.

A node $v$ of G is called *non-resolvable* if $S(G) = S(G \setminus \{v\})$. The intuition is that the removal of the node $v$ has no effect on the samples. On the other hand a node $v$ is called *resolvable* if $S(G) \neq S(G) \setminus \{v\}$. The idea is that no algorithm can detect a non-resolvable node considering only the samples of $G$ and $G'$, while some algorithm may be able to detect a resolvable node.

Next we identify nodes in $G$ that determine the topology (as well as the branch lengths which represent the time in generations to the next coalescent event) in the $K$ marginal trees and can lead to a structure-preserving transformation.

**Definition 3.4** (structure preserving transformation)**.** Given two instances of the ARG, $G$ and $G'$, if each of the $K$ embedded trees in $G$ and $G'$ are identical in topology and branch lengths (in generations), then $G'$ preserves the structure of $G$ and vice versa.

**Definition 3.5** (t-coalescent node)**.** A coalescent node in $G$ is *t-coalescent* if and only if it is also a coalescent node in at least one of the $K$ embedded trees of $G$.

For ease of exposition, the *state* of a vertex refers to a node being a *genetic-exchange* node (recombination node), *t-coalescent* or *non t-coalescent*. Also, recall that a chain node is a coalescent node that has a single parent and a single child, thus a non t-coalescent node. Any node is in exactly one of these three states.

The following Theorems and Definitions have been presented in [27].

**Theorem 3.6.** *If $G' = G \setminus U$ and no t-coalescent vertex of $G$ is in $U$, then $G'$ is structure-preserving.*

**Theorem 3.7.** *A resolvable coalescent node $v$ is also t-coalescent in $G$.*

In other words, if a set of coalescent nodes $U$ that are not t-coalescent are removed from $G$ to obtain $G'$, then $G$ and $G'$ are structure preserving. These useful properties lead to the definition of a core of an ARG, i.e., an its compact version which is structure and sample preserving. This is called minimal descriptor (introduced in [15]). In other words, a minimal descriptor is a non-redundant structure that can be extracted from any ARG.

**Definition 3.8** (minimal descriptor)**.** An ARG $G$ is a minimal descriptor if and only if every coalescent vertex, except the GMRCA, is t-coalescent.

Another equivalent definition of minimal descriptor is the following.

**Definition 3.9** (minimal descriptor)**.** An ARG $G_{md}$ is a minimal descriptor of $G$ if and only if (a) $G_{md}$ is a minimal descriptor, (b) $G_{md}$ preserves the structure of $G$, and (c) $G$ and $G_{md}$ are samples preserving, i.e., $S(G) = S(G_{md})$.

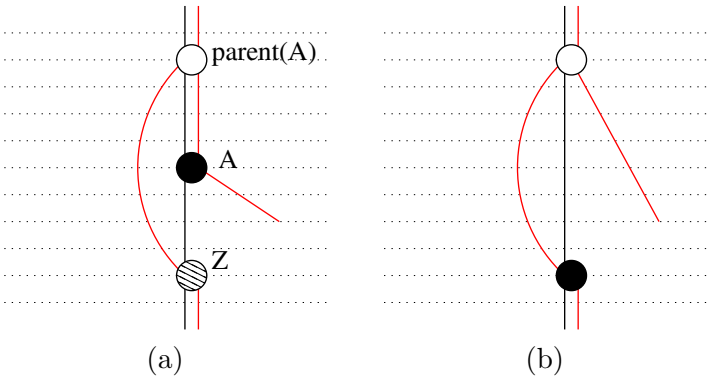



FIGURE 3.7: Portions of a graph: A solid black node is a non t-coalescent node; a hatched node is a genetic-exchange node, while a white node can be of any one of the three states. (a) A non t-coalescent node $A$ with its parent and genetic exchange child $Z$. The colors $l$ and $l'$ of Lemma 3.13 are red and black respectively. (b) shows the removal of the non t-coalescent node $A$ that changes the state of $Z$ to non t-coalescent node.

The minimal descriptor $G_{md}$ of an ARG $G$ is biologically and evolutionarily relevant as it is structure preserving, meaning that the embedded (marginal) trees of $G$ and $G_{md}$ are identical and it is samples preserving, meaning that the allele statistics (including allele frequencies) in the samples in both $G$ and $G_{md}$ are identical.

**Lemma 3.10.** *Removing a non t-coalescent node $A$ may affect the state of only its immediate neighbors, i.e., the parent or the children of $A$.*

**Lemma 3.11.** *Let $A$ be a non t-coalescent node. Then removal of $A$ does not change the state of its parent but may change the state of its genetic-exchange children.*

**Corollary 3.12.** *Removal of a t-coalescent node can only change the state of genetic-exchange nodes of the ARG and this state is never restored by further removal of subsequent non t-coalescent nodes.*

Note that in our discussion $l$ would represent one of the colors (say red) and the $l$-path is a path of red edges. Let $s = sg(v_0) \cap sg(v_1) \cap ... \cap sg(v_d)$. Then if $l \in s$, then an $l$-path of length $d$ on the graph is the directed path $v_0$, $v_1$, $v_2$, .., $v_d$. On the other hand if $s$ is empty, then there exists no path from $v_0$ to $v_d$ (of any color). A scenario where a node changes state is the following.

**Lemma 3.13.** *Let $A$ be a non t-coalescent node. Let the unique parent of $A$ be denoted as parent($A$). Let $Z$ be the child of $A$ which is a genetic-exchange node. Then $Z$ will change state if and only if the following holds. There exists a color $l$ such that there is an $l$-colored edge between parent($A$) and $Z$; there is an $l$-colored edge between parent($A$) and $A$; and there exists a color $l' \neq l$ with an $l'$-colored edge between parent($A$) and $Z$.*

See Fig 3.7 for an example where a genetic exchange node changes state. Here the $l$-color is the red edge.

Now we are ready to present out proof on the uniqueness of the minimal descriptor of an ancestral recombination graph.

### 3.7.2 Reduced ARG (mdARG) is unique

To avoid any biases in the comparison study, we first prove the following:

**Lemma 3.14.** *Given an instance of an ARG, it can be reduced uniquely to its minimal descriptor.*

*Sketch of proof*: We give the following construction. Consider a genetic-exchange node $x$. Let $d > 0$ be the smallest possible number such that there exists some $l$-path from $A = v_0$, $v_1$, $v_2$, ...,$v_d = X$, where only $v_0$ in the path is a non t-coalescent node and the others (i.e. $v_k, 0 < k \leqslant d$) are not. Let $A_1$, $A_2$, ..., $A_h$ be $h$ such t-coalescent nodes. Then let $Lbl(X) = \{A_1, A_2, ..., A_h\}$ and let $dep(X) = d$. For every other node let the labels be empty. See Fig 3.8 for an example.

Each genetic-exchange node is labeled as above. Let iterations be defined as follows. At Iteration 0, all non $t$-coalescent nodes are removed. At Iteration $i + 1$, the nodes that became t-coalescent due to state change in Iteration $i$ are removed. Then we claim that at Iteration $i > 0$, if node $v$ is removed then, $dep(v)$ must be $i$.

The above says that at Iteration $i$, it is possible that a node $v$ with $dep(v) = i$, may not be removed. The precise conditions under which a state change occurs is as follows, which is a natural extension of Lemma 3.13.

**Lemma 3.15.** *Let $A$ be a non t-coalescent node. Let the unique parent of $A$ be denoted as parent($A$). Let $Z$ be the child of $A$ which is a genetic-exchange node with $A \in Lbl(Z)$ and $dep(Z) = d$. Then $Z$ will change state at Iteration $d$ if and only if the following holds. There exists a color $l$ such that there is an $l$-colored edge between parent($A$) and $Z$; there is an $l$-colored path between parent($A$) and another parent $Z$ (other than the node parent($A$)); and there exists a color $l' \neq l$ with an $l'$-colored path between parent($A$) and $Z$.*

Further, no other genetic-exchange node can change state. This ends the proof of the theorem. □

**Corollary 3.16.** *An ARG is reduced to an mdARG in no more than $K$ iterations, where $K$ is the number of marginal trees.*

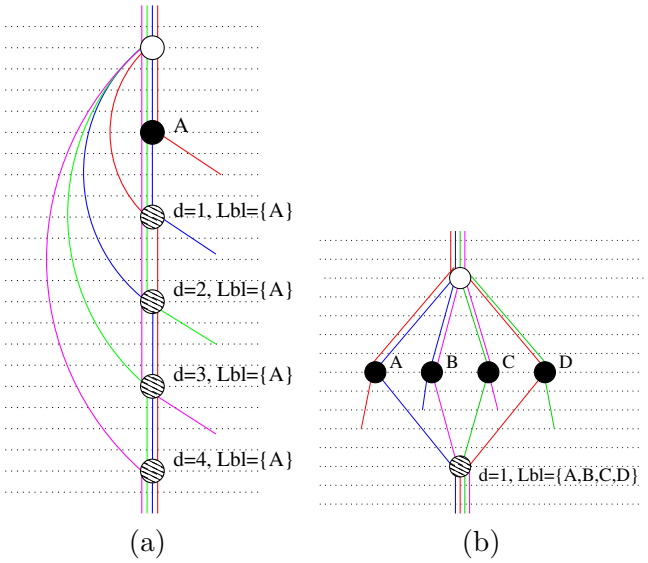



FIGURE 3.8: Portions of a graph: A solid black node is a non t-coalescent node; a hatched node is a genetic-exchange node, while a white node can be of any of the three states. (a) The non t-coalescent node is marked $A$. (b) The non t-coalescent nodes are marked $A$, $B$, $C$ and $D$.

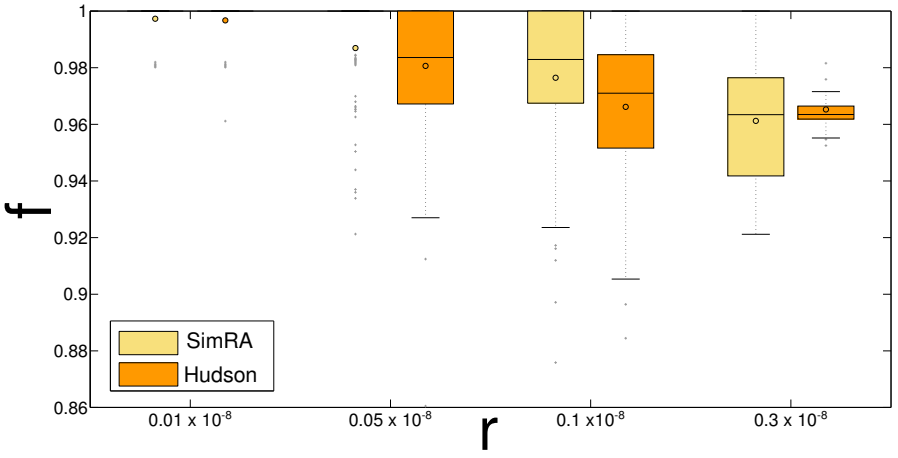



FIGURE 3.9: The box-and-whisker diagrams of the ratio $f$, for different values of $r$. The other parameter values are the same as used in Fig 3.5 and for each parameter setting both the algorithms were run 100 times. Recall that higher the values of $f$, higher the compaction factor.

For a non t-coalescent node $A$, parent($A$) can have at most $K$ children that have distinct color edges between them and parent($A$). Hence no changes occur to the graph after $K$ iterations. Fig 3.8 (a) gives an example.

Lemma 3.14 proves that the order of the operations (i.e., deletion of redundant nodes) is immaterial and it leads to the same non-redundant core.

### 3.7.3 Comparing the compactness of SimRA and Hudson

In order to compare the compactness of ARGs produced by SimRA and Hudson we run both the algorithms one hundred times for different parameter sets-up that differ in recombination rate values and we computed the minimal descriptor of each resulting ARG. As compactness measure we use $f$, the ratio of the number of nodes in the minimal descriptor to the number of nodes in the original ARG, as used in [28]. Observe that the closer the value of $f$ to 1, the less redundant is the ARG and thus more compact. Fig 3.9 shows that the ARGs produced by SimRA are systematically more compact than the ones produced by Hudson algorithm. Moreover, Hudson algorithm had to be aborted and re-run several times and it took over six months just to complete the comparison study shown in Fig 3.9 while SimRA was done with the four hundred runs in less than half a day. Hence, the superior performance of SimRA, as shown in Section 3.6 is possibly due to the fact that SimRA's ARGs are much more compact (see Section 3.7 for a rigorous definition of compactness).

## 3.8 On the uniqueness of GMRCA

In the following we give a prove on the uniqueness of GMRCA.

Let $\Omega$ denote the set of all (infinite) graphs, with nodes partitioned into distinct levels, or generations, with $N$ nodes at each level, and each node having no more than two parents. For each $X \in \Omega$, and any subset $V$ of the nodes at level 0, there is an induced subgraph of $X$, namely the ARG induced by $V$ and we call this the ARG associated with $X$.

Following [22] we introduce a probability measure on $\Omega$ as follows.

**Definition 3.17** (Truncation)**.** For $X \in \Omega$ and $h > 0$, we denote by $X_h$ the *truncation* of $X$ to depth $h$, i.e., $X_h$ is the finite induced graph from $X$ on the set of vertices of level $\leqslant h$.

Similarly, for a subset $E \subset \Omega$, and $h > 0$, we denote $E_h = \{X_h \mid X \in E\}$. We say that $E$ is *finitely determined* if there exists some $h_0$, such that $X \in E \Leftrightarrow X_{h_0} \in E_{h_0}$, and in this case we denote $\mu(E) = \frac{|E_{h_0}|}{|\Omega_{h_0}|}$.

The family, $\mathcal{F}$, of finitely determined subsets $E \in 2^{\Omega}$ clearly forms a field, and thus by the Caratheodory extension theorem (see for example [29], Theorem 1.1, page 4), $\mu$ can be uniquely extended to the $\sigma$-field generated $\mathcal{B}$ by this family. We denote this measure also by $\mu$ and consider $\Omega$ as a probability space with measure $\mu$.

Let $E^{\text{unq}} \subset \Omega$ be the set of graphs $X \in \Omega$, such that the ARG associated to $X$ has a unique GMRCA. The following theorem follows from the definition of the measure $\mu$. It assures us that almost every ARG has a unique GMRCA. In fact, in over ten thousand simulations, of which about three thousand are reported in this paper, SimRA terminated in every instance with a unique GMRCA.

**Theorem 3.18.** *The subset $E^{\text{unq}}$ is measurable and $\mu(E^{\text{unq}}) = 1$.*

*Proof.* For $h > 0$, let $E_h^{\text{unq}}$ denote the subset of ARGs having a unique GMRCA at level $\leqslant h$. Then, $E_h^{\text{unq}}$ is finitely determined and hence measurable, and

$$E^{\text{unq}} = \cup_{h>0} E_h^{\text{unq}},$$

being a countable union of measurable sets is measurable as well.

We now prove that $\mu(E^{\text{unq}}) = 1$. Suppose that the list of nucleotides in the chromosome is $\alpha_1, \ldots, \alpha_N$. For $1 \leqslant i \leqslant N$. Let $X$ be a random ARG, and for $1 \leqslant i \leqslant N$, let $X_i$ be the marginal forest corresponding to the nucleotide $\alpha_i$. Let for $h > 0$, $n_{i,h}(X)$ denote the number of vertices in $X_i$ at level $h$. It is easy to see that for every $h > 0$, $Pr(n_{i,h+1}(X) < n_{i,h}(X))$ is positive and bounded away from 0 by a constant independent of $h$. This implies that with probability one, $X_i$ is a tree. If $X_i \subset X$ is a tree, we let $\Gamma_i$ be the infinite path starting at the last node with positive in-degree in $X_i$ and extending to infinite height. Such a path exists for all $i, 1 \leqslant i \leqslant N$ with probability one, and it is also easy to see that with probablity one the paths $\Gamma_i, 1 \leqslant i \leqslant N$ will intersect at a common vertex of the ARG $X$. Such a vertex with the least height is the unique GMRCA of $X$, using Theorem 1, and we have proved that it exists with probability one. □

**Corollary 3.19.** *The measure of the space of all ARGs with no unique GMRCA is zero.*

## 3.9 Four Quantitative Hallmarks of ARG network

In this section we identify some quantitive hallmarks or characteristics of an ancestral recombination graph and we derive approximations of the expected hallmark values as closed-form functions of the ARG parameters.

Indeed, the ARG is a random object defined by the following parameters:

- $m$: the extant sample size;
- $g$: the length of the genomic segment whose common history is being tracked;

- $N$: the population size;
- $r$: the recombination rate;
- $\mu$: the SNP mutation rate.

Moreover, other polymorphisms, such as STRs, can be incorporated just as the SNP mutations are.

Note that a backward simulator can also accommodate multiple founding ancestors $m'$. While the number of surviving lineages does not go down monotonically from $m$ to 1, it is reasonable to impose that a truncated ARG has $m' < m$ active lineages or roots. Note that if $m' = 1$, then the lone active lineage is the GMRCA. Note that the unique founding ancestor, GMRCA, is attained in an ARG with probability 1 (see Section 3.8).

We consider the following four quantities as the hallmarks of the random object ARG with parameters $m$, $N$, $g$, $r$, $\mu$:

1. Depth of the ARG;
2. Number of non-mixing segments in the sample population;
3. Number of polymorphic sites in the sample population;
4. Diversity (defined by the polymorphisms) in the sample population.

In the following we present how we derive closed-form functions of the ARG parameters that provide approximations of the expected hallmark values. We did not find analytic or closed-forms of the expected values for the general scenario in literature, except some very specialized cases such as depth of GMRCA in the absence of recombinations [10]. Our derivations are based on the theorems and observations in [22].

Indeed, we find that if we require a single population only to study the hallmark expected values, but not the sample population, then the closed form approximations are tight enough to make the actual simulation redundant.

### 3.9.1 Closed-Form approximations of the expected hallmark values

First, we need to define two new notions: the *depth* of a node and the *girth* of an edge.

**Definition 3.20** (depth of a node)**.** An edge length, as well as *depth* of a node, is defined to be in time units. The unit of time is measured in generations. The depth of each node is measured from the leaf nodes and the depth of a leaf node is defined to be 0.

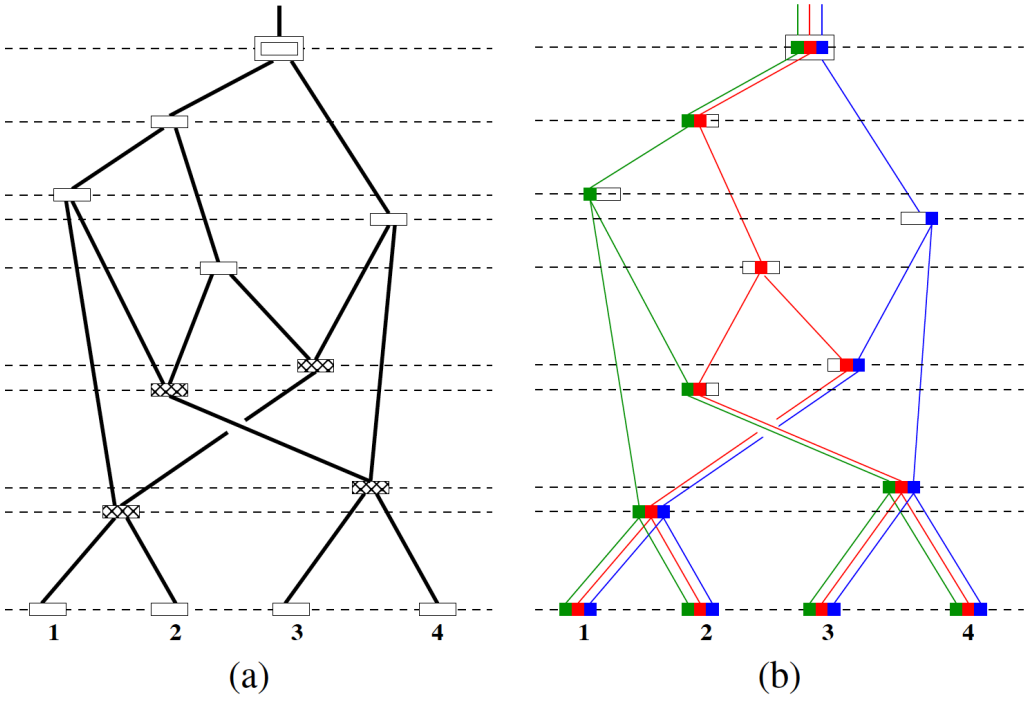



FIGURE 3.10: The horizontal lines denote the time at which an event (coalescence or recombination) occurs. (a) An example of an ARG with recombinations shown as hatched nodes. (b) shows the flow of the 3 non-mixing segments. These are shown in three distinct colors, green, red and blue (from left to right on the chromosomal segment).

**Definition 3.21** (girth of an edge)**.** The *girth* of an edge is defined to be the product of the edge length and the genomic segment width in the edge annotation (see Observation 1).

Moreover consider the following observations:

1. the ARG network is decomposed into overlapping trees (see Theorem 3.22);
2. for each tree, we compute the depth of each node and the girth of each edge, using Kingman coalescence. The depth of a tree is simply the depth of its root node. The girth of the tree is the sum of the girth of each edge of the tree;
3. the depth and girth of each tree is used for approximating the ARG hallmark values. However, the interdependence of the trees complicates these computations (see details below).

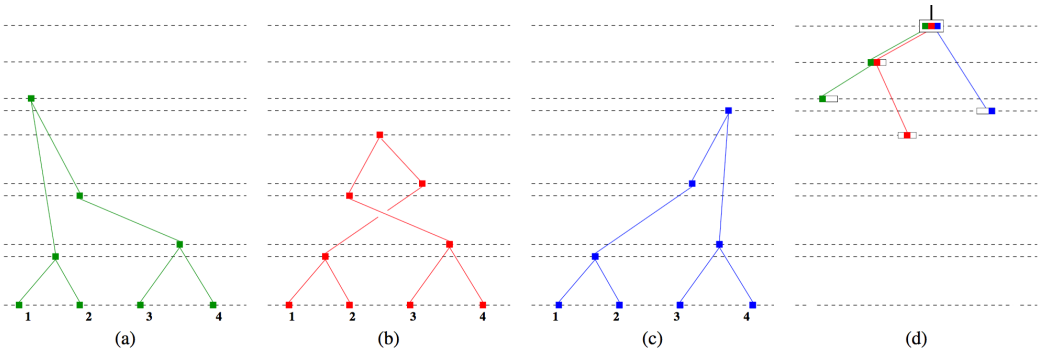



FIGURE 3.11: The decomposition of the ARG of Fig. 3.10 into four trees of Eqn. (3.27): $T_1$, $T_2$ and $T3$ corresponding to the three non-mixing segments are shown in (a)-(c) and $T'$ is shown in (d)

### Mathematical details

To keep this section self-contained, we report here the following basic identities. Let $0 < m' < m$.

$$\sum_{i=m'+1}^{m} \left( \frac{1}{i-1} - \frac{1}{i} \right) = \frac{1}{m'} - \frac{1}{m}, \tag{3.19}$$

$$\sum_{i=2}^{m} \frac{1}{i-1} = 1 + \frac{1}{2} + \ldots + \frac{1}{m-1} \approx \log m, \tag{3.20}$$

$$\sum_{i=m'+1}^{m} \frac{1}{i-1} \approx \log m - \log m' = \log \frac{m}{m'}. \tag{3.21}$$

Consider a tree with $m$ leafnodes. Using Kingman coalesence, all the non-leaf nodes of the tree can be written in increasing depth (from the leafnodes) as $v_1$, $v_2$, .., $v_{m-1}$, with the active lineages decreasing by one at each node. Let $t_i$ denote the depth of $v_i$ from $v_{i-1}$ where depth of $v_0$ is defined to be 0. Then the tree truncated at a depth that has $m'$ active lineages, is written as $T_{m,m'}$. Let $H_{T_{m,m'}}$ be the depth of this tree. Then using Property 3, linearity of expectations, and the above identities we get:

$$\begin{aligned} \mathbb{E}(H_{T_{m,m'}}) &= \sum_{i=1+m'}^{m} \mathbb{E}(t_i) = \sum_{i=1+m'}^{m} \frac{1}{\binom{i}{2}} = \sum_{i=1+m'}^{m} \frac{1}{\frac{i(i-1)}{2}} = \\ &= 2 \sum_{i=1+m'}^{m} \left( \frac{1}{i-1} - \frac{1}{i} \right) = 2 \left( \frac{1}{m'} - \frac{1}{m} \right). \end{aligned} \tag{3.22}$$

Let $g$ the length of the genomic segment carried by each edge in the tree and the girth of the tree be $wt_{T_{m,m'}}$. Then by using 3.21 we obtain:

$$\begin{aligned}
\mathbb{E}(wt_{T_{m,m'}}) &= \sum_{i=1+m'}^{m} i\mathbb{E}(t_i)g \\
&= g\sum_{i=1+m'}^{m} \frac{i}{\binom{i}{2}} = 2g\sum_{i=1+m'}^{m} \frac{i}{i(i-1)} = 2g\sum_{i=1+m'}^{m} \frac{1}{i-1} \qquad (3.23)\\
&\approx 2g\log\frac{m}{m'}. \qquad (3.24)
\end{aligned}$$

The complete tree with a single root node is written as $T_{m,1}$ and

$$\begin{aligned}
\mathbb{E}(H_{T_{m,1}}) &= 2\left(1-\frac{1}{m}\right), \qquad (3.25)\\
\mathbb{E}(wt_{T_{m,1}}) &= 2g\log m. \qquad (3.26)
\end{aligned}$$

We recall the following from [22] relating population genetics entities with graph entities like least common ancestor (LCA). A *non-mixing* genetic segment does not have any recombination event in the common history of the $m$ samples.

**Theorem 3.22.** *Let $\mathcal{G}$ be an ARG with some $K \geqslant 1$ non-mixing segments. Then $K$ marginal trees are embedded in $\mathcal{G}$ and the GMRCA of $\mathcal{G}$ is the LCA of the $K$ LCAs of the $K$ marginal trees.*

Fig 3.6 gives a simple illustration on an ARG on four samples with three non-mixing segments. An alternative view of the theorem is as follows: Let $\mathcal{T}(l,b)$ denote a tree defined on $l$ leaf nodes each carrying the segment of length $b$. Then for some partition of genome segment $g$ into $K$ non-overlapping segments, where $g = g_1 \cup g_2 \cup ... \cup g_K$,

$$\mathcal{G} \approx \left(\bigcup_{k=1}^{K} \mathcal{T}_k(m, g_k)\right) \cup \mathcal{T}'(K, g), \qquad (3.27)$$

where the roots of the $\mathcal{T}_k$'s are the leaves of $\mathcal{T}'$. In the example illustrated in Fig. 3.10 and Fig. 3.11; $T_1$, $T_2$ and $T_3$ are as in Fig. 3.11(a)-(c) and $T'$ as in (d).

**Corollary 3.23.** *If $H_1$ is the maximum of the depths of $T_k(m, g_k)$ and $H_2$ is the depth of $\mathcal{T}'(K, g)$ and $H$ is the depth of the GMRCA of the $\mathcal{G}$, then*

$$H = H_1 + H_2. \qquad (3.28)$$

**Corollary 3.24.** *The girth of ARG $\mathcal{G}$ is the sum of the girth of each $\mathcal{T}_k(m, g_k)$ and $\mathcal{T}'(K, g)$.*

Now we are ready to introduce the derivation of closed-form functions for the four quantitive hallmarks of an ARG.

### Expected number of recombinations $Z$

Consider $K$ from Theorem 3.22, $H_1$ and $H_2$ from Corollary 3.23. Then based on Eqn 3.25 and Corollary 3.23:

$$\begin{aligned} H_1 &= 2\left(1-\frac{1}{m}\right), \\ H_2 &= 2\left(1-\frac{1}{K}\right). \end{aligned} \tag{3.29}$$

The number of generations to the top of the ARG is $HN$. We assume infinite sites model of the recombination sites, i.e., a recombination occurs at a site no more than once. Also, notice that a recombination occurs only at the nodes of the ARG, but not along the edges, hence we use the following approximation of $Z$:

$$Z \approx (H_1 + H_2)Nrg. \tag{3.30}$$

We postpone the derivation of $\mathbb{E}(H)$ to the next section, but note that

$$\mathbb{E}(Z) \approx \mathbb{E}(H)Nrg.$$

### Expected depth of ARG $H$

Let $\alpha = Nrg$ and $\beta = \alpha H_1 + 1$. Since the number of un-mixing units $K$ (that is the number of marginal trees) is equal to the number of recombinations plus one, then using Eqn 3.30

$$K = Z + 1 = Nrg(H_2 + H_1) + 1 = \alpha(H_2 + H1) + 1 = \alpha H_2 + \beta. \tag{3.31}$$

Based on Eqns 3.29 and 3.31, we get

$$\alpha H_2^2 + (\beta - 2\alpha)H_2 + 2(1-\beta) = 0. \tag{3.32}$$

*Sketch of the proof.*

$$\begin{aligned}
H_2 &= 2\left(1-\frac{1}{K}\right), \\
H_2 &= 2\left(1-\frac{1}{\alpha H_2+\beta}\right), \\
H_2 &= 2\left(\frac{\alpha H_2+\beta-1}{\alpha H_2+\beta}\right), \\
(\alpha H_2+\beta)H_2 &= 2(\alpha H_2+\beta-1), \\
\alpha H_2^2+\beta H_2 &= 2\alpha H_2+2\beta-2, \\
\alpha H_2^2+\beta H_2-2\alpha H_2-2\beta+2 &= 0, \\
\alpha H_2^2+(\beta-2\alpha)H_2+2(1-\beta) &= 0.
\end{aligned}$$

As $\beta > 1$ and $\alpha > 0$, $(1-\beta) < 0$. Hence one of the roots is positive while the other is negative and we are interested only in the positive root. Thus we consider the larger of the two roots given below:

$$H_2 = \frac{2\alpha-\beta \pm \sqrt{(\beta-2\alpha)^2-8\alpha(1-\beta)}}{2\alpha}.$$

Recall that

$$H_1 = 2\left(1-\frac{1}{m}\right),$$

then

$$\mathbb{E}(H) \approx H_1 + H_2. \tag{3.33}$$

**Expected number of mutations $Y$**

Unlike recombinations, polymorphisms such as point mutations or STR mutations occur along the edge of the ARG. Thus an edge of length $t$ is $tN$ in units of generation and has $tN\mu$ mutations. Thus the girth of the network offers a better handle in computing the total number of polymorphisms. Here too we assume an infinite sites model, i.e., a mutation occurs at one site no more than once.

Next, Corollary 3.24 suggests that the girth of an ARG can be computed from the girth of the $K+1$ tree topologies. Then by Eqn 3.26,

$$\begin{aligned}
\mathbb{E}(wt_{\mathcal{G}}) &= \sum_{k=1}^{K} 2g_k\mathbb{E}(wt_{\mathcal{T}_{k_{m,1}}}) + 2g\mathbb{E}(wt_{\mathcal{T}_{K,1}}) \\
&= 2g\left(\log m + \log K\right).
\end{aligned}$$

Recall that $K = Z + 1$, then

$$\begin{aligned} E(Y) &\approx \mu N \mathbb{E}(wt_{\mathcal{G}}), \\ E(Y) &\approx \mu N 2g \left(\log m + \log K\right). \end{aligned} \tag{3.34}$$

**Expected diversity $D$ in the population samples**

Let $p_j$ be allele frequency of SNP $j$, i.e., if $a_j$ is the number of samples with a fixed SNP value then $p_j = a_j/m$. Then

$$D = \sum_j d_j \quad \text{where } d_j = p_j(1 - p_j).$$

To compute the approximation, we make the following assumptions: (a) Firstly, we count the polymorphisms (SNPs) only in $\mathcal{T}_k$ of Eqn 3.27 but not in $\mathcal{T}'$, since the latter potentially propagates the mutations to the entire sample. (b) Secondly, the contribution of each $\mathcal{T}_k$ to $D$ is the same on an average. (c) Finally, at each iteration $i$ of the girth computation the number of samples are partitioned equally, on an average, in the $\binom{i}{2}$ branches of each $\mathcal{T}_k$.

We augment the girth computation of Eqn 3.23 for each $\mathcal{T}_k$ as follows. At each iteration $i$, the girth of the tree $T_k$ is

$$\begin{aligned} i\mathbb{E}(t_i)g_k &= g_k i \left(\frac{1}{\binom{i}{2}}\right), \\ i\mathbb{E}(t_i)g_k &= g_k i \left(\frac{1}{\frac{i(i-1)}{2}}\right). \end{aligned}$$

Then the expected number of mutations at each iteration $i$ is

$$N\mu \frac{2g_k}{i-1},$$

and the $p_j$ of mutation $j$, based on assumption (c) above, is

$$p_j = \frac{m/i}{m} = \frac{1}{i}, \text{ giving } d_j = \left(\frac{1}{i}\right)\left(1 - \frac{1}{i}\right) = \frac{i-1}{i^2}.$$

Summing over all the mutations at iteration $i$:

$$N\mu\frac{2g_k}{i-1}d_j = N\mu\frac{2g_k}{i^2}.$$

Summing over all the iterations $i$ in $\mathcal{T}_k$ gives $2g_kN\mu\sum_{i=2}^{m}\frac{1}{i^2}$. Finally, based on assumption (b), we sum the contributions from all the $\mathcal{T}_k$. Thus

$$\mathbb{E}(D) \approx 2gN\mu\sum_{i=2}^{m}\frac{1}{i^2}. \tag{3.35}$$

**Expected height of a truncated ARG**

For a meaningful simulation of the intermixing populations around a scaffold, the single population in an edge should not be allowed to coalesce into a single lineage. Hence we study the truncated ARG. The lone founding ancestor, GMRCA, is attained in an ARG with probability 1 (see Section 3.8).

A backward simulator can also accommodate multiple founding ancestors $m'$, by stopping the simulation and truncating the ARG when there are $m'$ active lineages. While the number of surviving lineages does not go down monotonically from $m$ to 1, it is reasonable to impose that a truncated ARG has $m' < m$ active lineages. Also, it is reasonable to impose that $1 \leqslant m' << m/2$, to allow enough room for time to leave its footprint, in terms of mutations, in the population. Note that if $m' = 1$, then the lone active lineage is the GMRCA.

Although Thm 3.22 is based on the complete ARG with the GMRCA, in the approximation below, we use the theorem in the context of the truncated ARG that has $1 \leqslant m' < m$ roots. Consider $H_1$ and $H_2$ from Corollary 3.23. Then based on Eqns 3.22 and 3.25 we make the approximation:

$$\begin{aligned} H_1 &= 2\left(1-\frac{1}{m}\right), \\ H_2 &= 2\left(\frac{1}{m'}-\frac{1}{K}\right). \end{aligned}$$

As before we get,

$$\alpha m' H_2^2 + (m'\beta - 2\alpha)H_2 + 2(m' - \beta) = 0,$$

and we consider only the positive root of the following:

$$H_2 = \frac{2\alpha - m'\beta \pm \sqrt{(m'\beta - 2\alpha)^2 - 8\alpha m'(m' - \beta)}}{2\alpha m'}.$$

### 3.9.2 Summary of closed-form formulations

Here we summarize the formulas that have been used to compute the accuracy of the ARG sampling algorithm. Recall that $H$ is the height of the ARG; $Z$ is the number of recombinations; $Y$ is the number of mutations; $D$ is the diversity of the population. The formulas are recapitulated here, for convenience. Let

$$\begin{aligned}
H_1 &= 2\left(1 - \frac{1}{m}\right), \\
\alpha &= Nrg, \\
\beta &= \alpha H_1 + 1, \\
\gamma &= 2\alpha - \beta, \\
H_2 &= \frac{\gamma + \sqrt{\gamma^2 - 8\alpha(1-\beta)}}{2\alpha}, \quad \text{when } r > 0, \\
K &= (H_1 + H_2)Nrg + 1.
\end{aligned}$$

Then the four (expected) hallmark values are:

$$\begin{aligned}
\mathbb{E}(\mathrm{H}) &\approx H_1 + H_2, \\
\mathbb{E}(Z) &\approx K - 1, \\
\mathbb{E}(Y) &\approx 2\mu g N \left(\log m + \log K\right), \\
\mathbb{E}(D) &\approx 2gN\mu \sum_{i=2}^{m} \frac{1}{i^2}.
\end{aligned}$$

Given that $\alpha$ and $\beta$ values are as above, the expected height of truncated ARG is:

$$\begin{aligned}
H_1 &= 2\left(1 - \frac{1}{m}\right), \\
\gamma &= 2\alpha - m'\beta, \\
H_2 &= \frac{\gamma \pm \sqrt{\gamma^2 - 8\alpha m'(m' - \beta)}}{2\alpha m'}, \quad \text{when } r > 0, \\
K &= (H_1 + H_2)Nrg + 1, \\
\mathbb{E}(\mathrm{H}) &\approx H_1 + H_2.
\end{aligned}$$

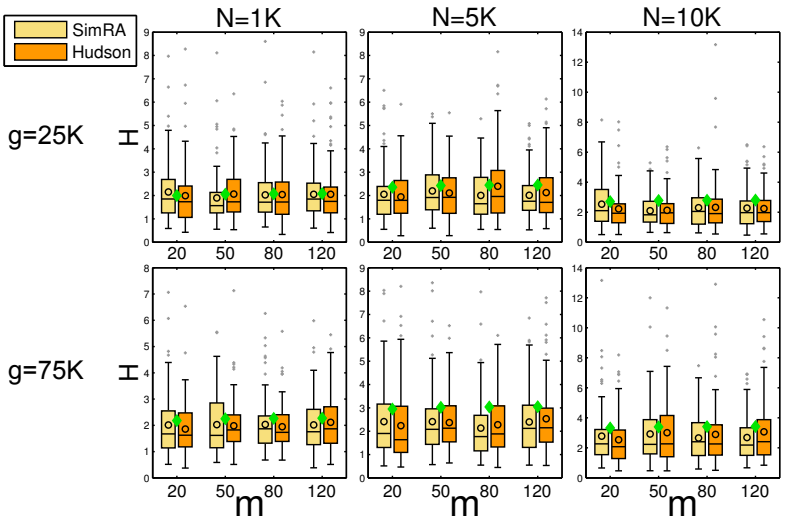



FIGURE 3.12: The closed form expected values (green diamonds) of the height $H$ of ARGs are compared against empirical values for different parameter values of $m$, $g$, and $N$. The recombination rate used is $r = 0.1 \times 10^{-8}$ Morgan/bp/generation, the mutation rate is $\mu = 1.5 \times 10^{-8}$ mutations/bp/generation and $m' = 1$.

## 3.10 Accuracy

We present the accuracy of SimRA by comparing the four hallmark values to the ones computed by Hudson. The results are shown in Figs. 3.12, 3.13, 3.14, 3.15. The closed form expected values are compared against empirical values for different parameter values of sample size $m$, segment length $g$, population size $N$. We run both the algorithms, SimRA and Hudson, 100 times for each parameter set-up. The recombination rate used is $r = 0.1 \times 10^{-8}$ Morgan/bp/generation and $m' = 1$. Note that the mutation rate affects only 3.14 and 3.15 and $\mu = 1.5 \times 10^{-8}$ mutations/bp/generation. The box-and-whisker diagram summarizes the results as in Fig 3.5. In each plot, the green diamond is the expected value as computed by the closed form, while the hollow circles are the observed empirical values by SimRA and Hudson.

Notice that not only SimRA and Hudson estimates are very similar, if not identical, over 100 runs for each parameter set-up, but the closed form is also a good approximation.

Moreover we used the closed-form function for the expected height of a truncated ARG to compare the estimates produced by both the algorithm SimRA and Hudson against the expected values. Fig 3.16 shows the expected height of truncated ARG, for different values of $m'$ as fractions of $m$, compared against the observed values obtained using both SimRA and Hudson. Again, note the tightness of the approximations.

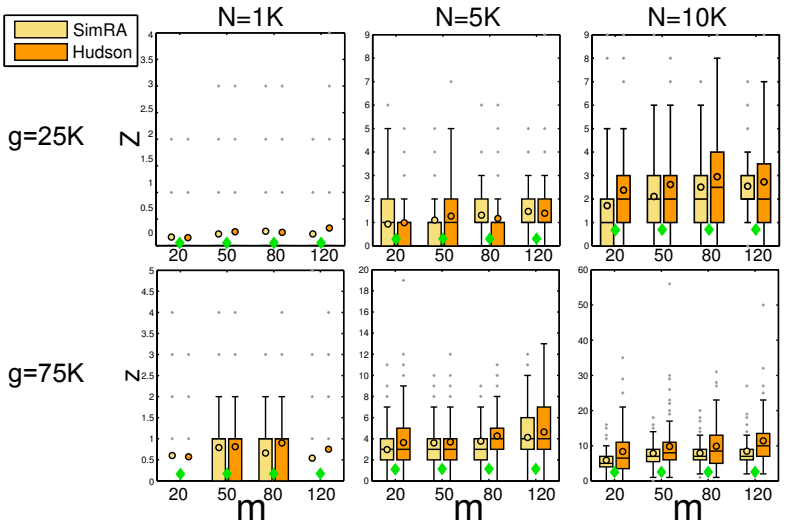



FIGURE 3.13: The closed form expected values (green diamonds) of the number of recombinations $Z$ in ARGs are compared against empirical values for different parameter values of $m$, $g$, and $N$. The recombination rate used is $r = 0.1 \times 10^{-8}$ Morgan/bp/generation, the mutation rate is $\mu = 1.5 \times 10^{-8}$ mutations/bp/generation and $m' = 1$.

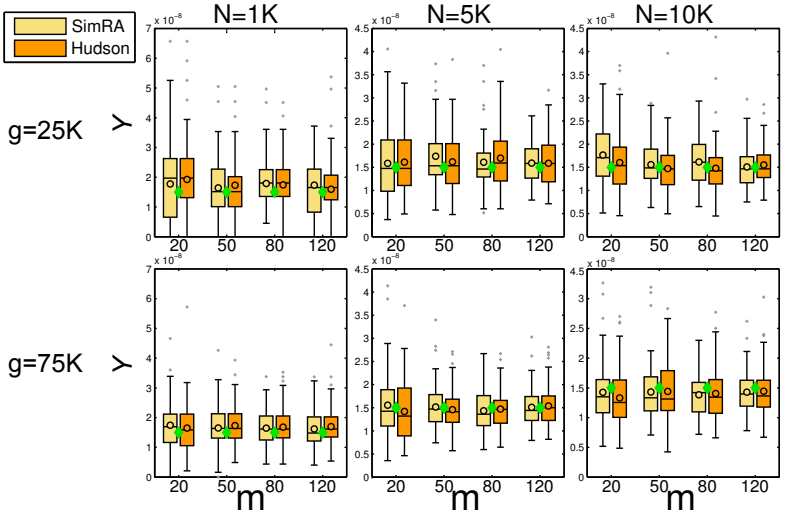



FIGURE 3.14: The closed form expected values (green diamonds) of the number of mutations $Y$ in ARGs are compared against empirical values for different parameter values of $m$, $g$, and $N$. The recombination rate used is $r = 0.1 \times 10^{-8}$ Morgan/bp/generation, the mutation rate is $\mu = 1.5 \times 10^{-8}$ mutations/bp/generation and $m' = 1$.

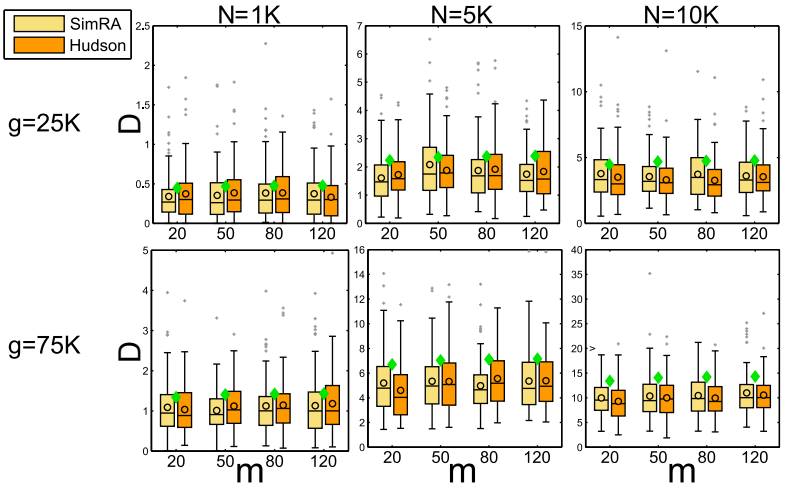



FIGURE 3.15: The closed form expected values (green diamonds) of the diversity of the population $D$ are compared against empirical values for different parameter values of $m$, $g$, and $N$. The recombination rate used is $r = 0.1 \times 10^{-8}$ Morgan/bp/generation, the mutation rate is $\mu = 1.5 \times 10^{-8}$ mutations/bp/generation and $m' = 1$.

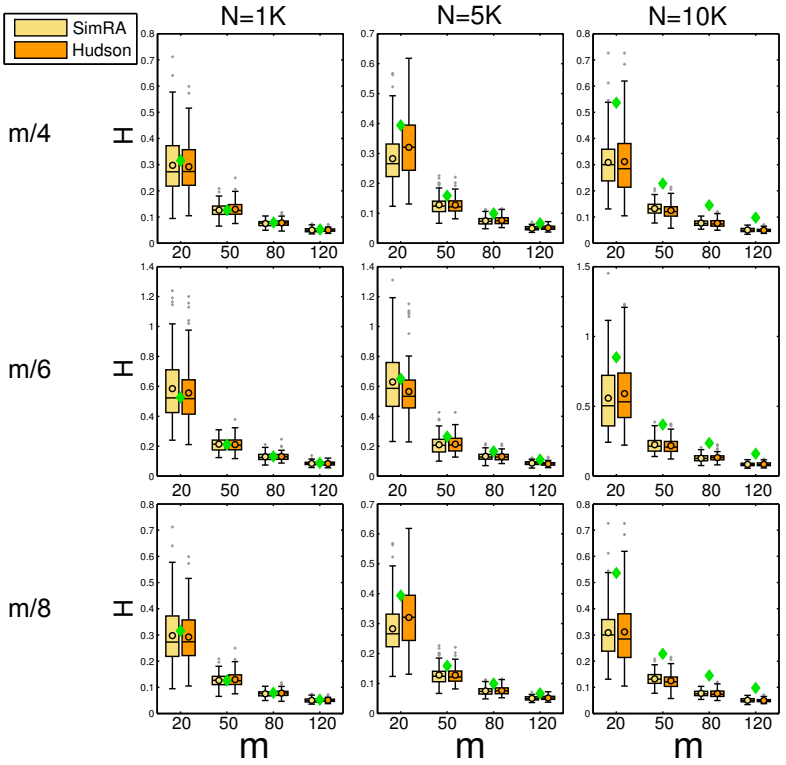



FIGURE 3.16: The height of the truncated ARG for different values of $m'$ as fractions of $m$. The recombination rate used was $r = 0.1 \times 10^{-8}$ Morgan/bp/generation and g=25K. Each parameter setting was run 100 times and the results are captured in the box-and-whisker diagram (as detailed in Fig 3.5). In each parameter setting, the green diamond is the expected value as computed by the closed form, while the hollow circles are the observed empirical values by SimRA and Hudson.

## 3.11 Using closed-form approximations for complex scenarios parameter set-up

In the following, we give an example of the use of the closed-form approximations to design appropriate parameter set-up for a given scaffold. In particular, the expected depth of a truncated ARG is used.

Suppose that a scaffold (see Fig. 3.17 as an example) and the common parameters for all edges (i.e., recombination rate, segment length and mutation rate) are given as input. Suppose also that the sample size of each contemporary population is given.

In order to design a parameter set-up for the given scaffold it is necessary to choose appropriate values for the time parameters and the effective population size for each edge in the scaffold. Hence, the closed form of the expecting depth of an ARG can be used, by applying the following procedure.

Each edge (or population) $e_j$ is characterized by the start time, $start_j$, when $e_j$ starts to be constructed during the backward simulation process, and end time $end_j$, when the ARG for $e_j$ stops growing and is involved in a merging or splitting event. Moreover, for each edge $e_j$, $l_b(e_j)$ represents the number of lineages at the bottom of $e_j$, and $l_t(e_j)$ the number of lineages at the top of $e_j$. Note that for each contemporary population the number of lineages at the bottom corresponds to the sample size.

Suppose the scaffold has $n$ event times, then their values are set in ascending order ($t_1$, $t_2$, $\ldots$, $t_n$) by computing the expected depths of the edges of the scaffold from the bottom to the top.

Consider a time $t_i$. If at time $t_i$ the edge (representing a population in the scaffold) $e_j$ is split, then an approximate value for $t_i$ can be computed by adding the start time of $e_j$, $start_j$ to the expected depth of $e_j$. $E(H_j)$ is computed by using the closed form function of the expected height of a truncated ARG (See Section 3.9.2). Note that if $e_j$ is a leaf population, its start time is known and it is equal to zero. Otherwise, if $e_j$ is a internal edge in the scaffold, then its start time corresponds to a previous time $t_k$, where $k < j$, that have been already set in a previous step of this procedure.

If $t_l$ is the time in which two populations $e_j$ and $e_k$ merge, then the expected depth of $e_j$, $E(H_j)$ plus the start time $start_j$ of $e_j$ has to be approximatively equal to the expected depth of $e_k$ plus the start time $start_k$ of $e_k$. In other words, the depths of the two paths that meet at time $t_l$, when $e_j$ and $e_k$ merge and form a new ancestral population $e_l$, have to be approximatively equal, so that it is possible to assign a consistent value to $t_l$. Note

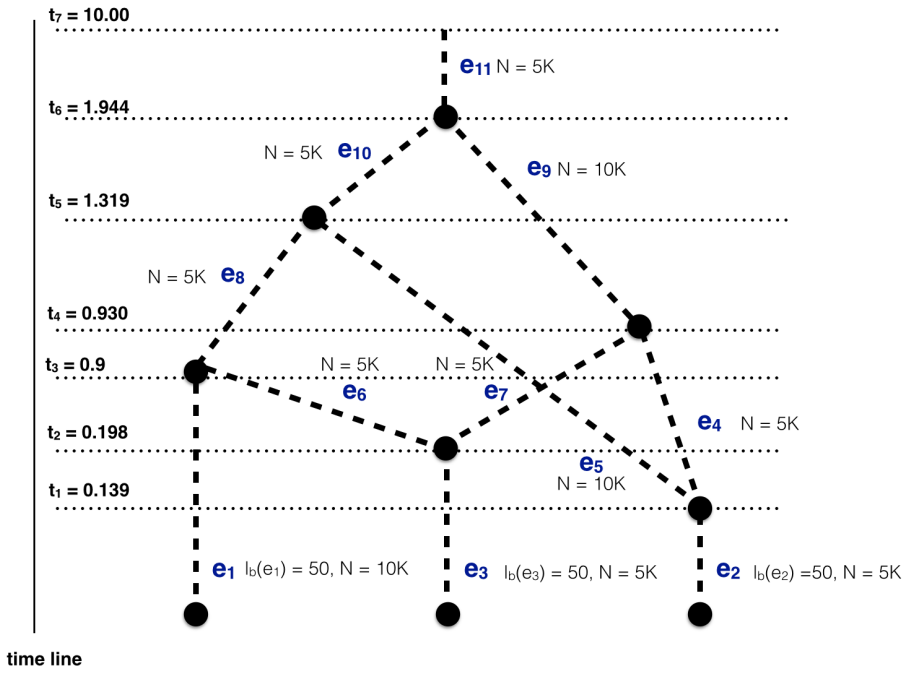



FIGURE 3.17: Scaffold with 3 contemporary populations where two of them are admixed. Each edge is labeled with the respective ID and effective population size. The leaf nodes are also labeled with the selected sample size. The figure shows the time line and the chosen values for the time parameters $t_1, t_2, \ldots, t_7$.

that $t_l$ will correspond to the start time of the population $e_l$ deriving from the merge of $e_j$ and $e_k$.

In order to obtain equal or closed values of depths of two paths involved in a merge event and to respect the ascending order of times, it may be necessary to adjust some parameters during the process of computing the expected depths of the edges. The parameters that can be adjusted are, for example, the effective population size $N_j$ and the number of surviving lineages at the top $l_t(e_j)$ of each edge $e_j$.

Indeed, while the recombination rate $r$, the length of the segment $g$ and the number of lineages at the bottom $l_b(e_j)$ of an edge $e_j$ are already given, it is possible to select appropriate values for the effective population size $N_j$ and the number of lineages at the top $l_t(e_j)$ to obtain an expected depth $E(H_{e_j})$ consistent with the constraints given by the scaffold.

Note that in the closed form $m$ is the number of lineages $l_b(e_j)$ at the bottom $e_j$, while $m'$ is the number of the lineages $l_t(e_j)$ at the top of $e_j$.

In the following a practical example of the above procedure is presented. Consider the scaffold in Fig. 3.17. Suppose that the following parameters are given in input: recombination rate $r = 0.1 \times 10^{-8}$, mutation rate $\mu = 1.5 \times 10^{-8}$, segment length

| | $N_j$ | $l_b(e_j)$ | $l_t(e_j)$ | $E(H_{e_j})$ |
|---|---|---|---|---|
| $e_1$ | 10000 | 50 | 15 | 0.898 |
| $e_2$ | 5000 | 50 | 25 | 0.139 |
| $e_3$ | 5000 | 50 | 20 | 0.169 |
| $e_4$ | 5000 | 12 | 4 | 0.790 |
| $e_5$ | 10000 | 13 | 5 | 1.180 |
| $e_6$ | 5000 | 10 | 4 | 0.729 |
| $e_7$ | 5000 | 10 | 4 | 0.729 |
| $e_8$ | 5000 | 19 | 8 | 0.412 |
| $e_9$ | 10000 | 8 | 5 | 1.013 |
| $e_{10}$ | 5000 | 13 | 5 | 0.626 |
| $e_{11}$ | 5000 | 10 | 1 | 2.829 |

TABLE 3.2: The first three columns are respectively the selected values for the effective population size, the number of lineages at the bottom and number of lineages at the top of each edge in the scaffold. The last column is the expected depth of each edge computed by using closed-form (see Section 3.9.2).

$g = 75 \times 10^3$. Moreover the sample size for each contemporary population is known: $m_{e_1} = 50$, $m_{e_2} = 50$ and $m_{e_3} = 50$.

The procedure starts computing the time $t_1$, when the surviving lineages of $e_2$ are split. The time $t_1$ corresponds to the expected depth of $e_2$. It is convenient to choose $N_1 = 5000$ and $x = l_t(e_2) = 25$, in order that the expected depth of $e_2$ is $E(H_{e_2}) = 0.139$, thus $t_1 = 0.139$.

At time $t_2$ the surviving lineages of $e_3$ are split, then $t_2$ corresponds to the expected depth of $e_3$. Since $t_2$ has to be greater than $t_1$ and the effective population size $N_3 = 5000$ is chosen to be equal to $N_2$, it is necessary to set $l_t(e_3) < l_t(e_2)$ so that $t_2 > t_1$. Indeed, if $l_t(e_3) = 20$ the expected depth of $e_3$ is equal to $E(H_{e_2}) = 0.9$, thus $t_2 = 0.9$.

At time $t_3$ the respective surviving lineages of $e_1$ and $e_6$ merged. In this case, in order to select an appropriate value for $t_3$, it is needed to obtain similar values of the expected depth of $e_1$ and the sum of $t_2$ and the expected depth of $e_6$. By selecting the following values $N_1 = 10000$, $l_t(e_1) = 15$, $N_6 = 5000$ and $l_t(e_6) = 4$, then the two resulting values ($E(H_{e_1}) = 0.898$ and $t_2 + E(H_{e_6}) = 0.92$) are close, thus $t_3$ can be set as an intermediate value, $t_3 = 0.9$.

At time $t_4$ the surviving lineages of $e_7$ and $e_4$ are merged. Applying the same method as before, values for $N_7$, $N_4$, $l_t(e_4)$ and $l_t(e_7)$ are assigned. The time for $t_4$ is picked comparing the following two values: $t_1 + E(H_{e_4}) = 0.139 + 0.79 = 0.929$ and $t2 + E(H7) = 0.198 + 0.729 = 0.927$. These two values are very close so that $t_4 = 0.93$.

The parameters values that have been chosen during this process are summarized in Table 3.2.

At time $t_5$ the surviving lineages of $e_8$ and $e_5$ merge. Following the same process as before, a value between $t_3 + E(H_{e_8}) = 0.9 + 0.412 = 1.319$ and $t_1 + E(H_{e_5}) = 0.139 + 1.18 = 1.319$ can be chosen. In this case the two values are exactly the same thus $t_5 = 1.319$. Furthermore, $t_6$ is approximatively equal to 1.944 since $t_5 + E(H_{e_10}) = 1.319 + 0.626 = 1.945$ and $t_4 + E(H_{e_9}) = 0.93 + 1.013 = 1.943$. Finally the expected depth of the most ancestral population $e_{11}$ is approximatively $E(H_{e_{11}}) = 2.829$. Therefore $t_7$ can be set to $t_6 + E(H_{e_{11}}) = 4.773$, but in this case a larger threshold of time for example $t_7 = 10.00$ can be chosen.

**Availability.** SimRA (Simulation based on Random graph Algorithms) source, executable, user manual and sample input-output sets are available for downloading at:

`https://github.com/ComputationalGenomics/SimRA`

# Chapter 4

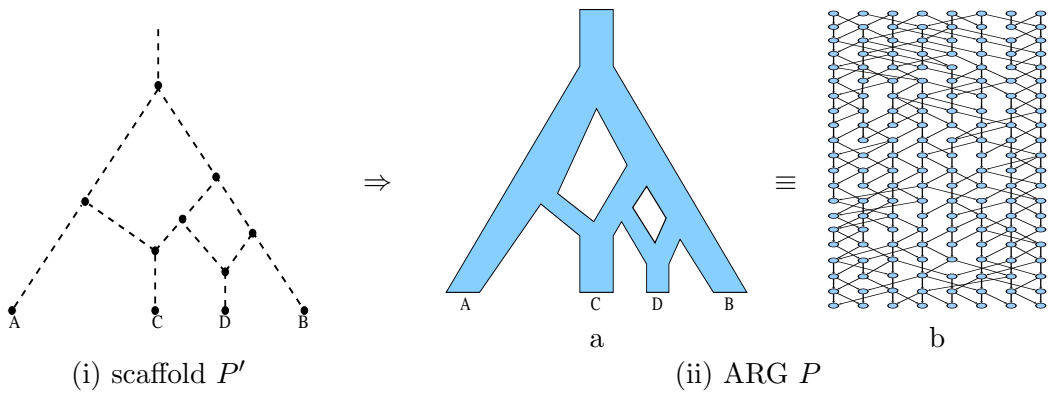
# Topological signatures for population admixture

In this chapter a new topological framework to detect admixture in contemporary related populations is introduced and applied to biological data. We illustrate, based on controlled simulations computed by SimRA, that the topological characteristics have the potential for detecting subtle admixture in related populations. Afterwards, we apply the technique successfully to a set of avocado germplasm data, proving that the approach has the potential for novel characterizations of relatedness in populations.

## 4.1 Motivation

Relatedness of populations is an interesting problem and has been studied extensively in the population genetics community [30, 31]. In the context of plant breeding, this understanding is very important in gauging the diversity in the genetic pool and using it effectively in breeding programs [32]. In the context of humans, admixture mapping of the genome is useful for disease or complex trait association studies [33, 34].

Various statistical models have been proposed in literature [31, 35] to characterize admixture which build mainly on linkage disequilibrium footprints via minimum allele frequencies of the markers. In [3] a combinatorial model based on persistence in topology to model and detect admixture in populations has been presented. The authors in [36] have used a similar model to study presence/absence of genetic exchange as recombination or reassortment in viral populations. However, the problem that we address here is a little more nuanced: to tease apart subtle admixtures from the usual interrelationships




(i) scaffold $P'$ (ii) ARG $P$

FIGURE 4.1: An example with four populations A, B, C, D. (i) shows the scaffold $P'$. (ii) shows a corresponding ARG $P$. (ii-a) shows the ARG with the "shape" of $P'$ superimposed on it, while the (ii-b) shows some of the details of $P$ of (ii-a). Note that in general the structure of $P'$ is not apparent from $P$ and the ARG $P$ simply looks like the one shown in (ii-b). See text for more details.

of related populations. In other words, the aim is to discern admixture from amongst the ubiquitous recombination events.

The problem is defined as follows.

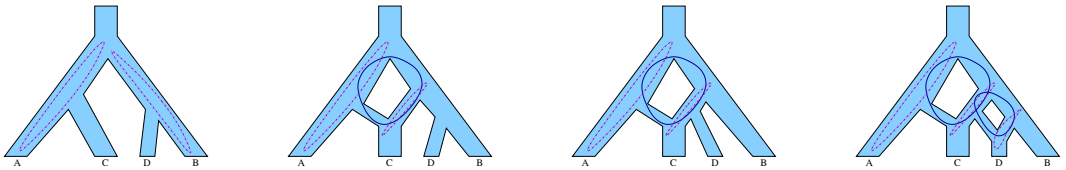
## 4.2 Problem Setting

Let $P$ be an ARG (or a subARG), where the leaf nodes have an additional population label. Fig 4.1 (ii) shows an example with four population labels. Let the relationship between the $m$ populations be defined by a DAG $P'$ with $m$ leaf nodes, called a scaffold, as shown in Fig 3.1 (i). See Section 3.2 for a detailed definition of scaffold.

We recall that each edge $e$ of $P'$ represents the evolution of a Wright Fisher population captured in a DAG say $P_e$. The union of each of these DAGs by appropriately gluing the ends of the edges corresponding to the nodes of $P'$ gives the ARG $P$ that can be written as:

$$P = \bigcup_{e \in P'} P_e.$$

Such a $P$ is shown in Fig 4.1 (ii), where the leaf nodes correspond to extant units of each population of $P'$; (ii-b) shows some of the typical details of enclosed area of (ii-a). Each row in (ii-b) is a generation and the edges denote the flow of genetic material towards the extant units at the leaf nodes (the arrows are not shown to avoid clutter). A node with two incoming edges in (ii-b) denotes a genetic exchange event such as recombination. We refer the reader to Section 2.7 for further details of a typical ARG $P$.




0 CP 1 CP 1 CP 2 CP

FIGURE 4.2: Examples of CPs shown as solid dark closed paths in the respective ARGs. In contrast, the dashed closed paths cannot correspond to CPs in the underlying scaffolds.

We say that a scaffold $P'$ defines *admixture* if there exists a closed path (CP) in $P'$. Moreover, observe that a recombination event in the evolution process leads to the occurrence of a CP in $P$.

Now we are ready to define the central problem as a riddle with three actors as follows.

**Problem 1.** *Tom generates a scaffold $P'$ on $m$ populations with the three parameters $len(e)$, $l_b(e)$ and $l_t(e)$ for each edge $e \in P'$ satisfying the junction constraints. Based on $P'$, Dick constructs an ARG $P$ on $m$ populations. Can Harry detect whether Tom's $P'$ has any CPs i.e., admits admixture, based on the data given to him by Dick:*

*Scenario I: the ARG $P$;*

*Scenario II: a subARG of $P$ that has all leaf nodes of $P$;*

*Scenario III: only the leaf nodes of $P$.*

#### Outline of our approach to the solution

Note that given an ARG or subARG $P$, its underlying scaffold $P'$ is not immediately computable. Indeed, due to recombinations, many CPs exist in $P$, but they do not necessarily indicate a CP (admixture) in $P'$. Fig 4.2 shows some examples. In [3] , we resort to topology and translate this problem into persistence homology computation in the Vietoris-Rips complex defined by $P$. Notice that Scenario I is an ideal situation while Scenarios II and III correspond to practical situations, and, we focus on the latter.

## 4.3 Topology Model

In this section we present a theoretical model that explains the topological signal for the presence or absence of admixtures in the populations being studied in the persistence diagrams that we compute. We model Scenario III of the last section as follows. Denote the leaf nodes of $P$, by $L(P)$. There exists a notion of distance between nodes $v, v'$ of

$L(P)$, denoted $w(v, v')$, obtained by setting

$$w(v, v') = \min_{u \in \mathrm{lca}(v,v')} \mathrm{depth}(u),$$

where $\mathrm{depth}(u)$ denotes the depth of the node $u$ in $P$ (measured in terms of the number of generations), the depth of any leaf node of $P$ is 0 and $\mathrm{lca}(v, v')$ is the set of least common ancestors of $v$ and $v'$ in $P$. Recall that the population labels of the leaf nodes partitions $L(P)$ into disjoint subsets, where each subset corresponds to a population. Let the set of populations be denoted by $\tilde{L}(P)$. Hence, there exists a surjective map, $\phi : L(P) \to \tilde{L}(P)$. The distance function $w(\cdot, \cdot)$ on $L$, induces a distance function $\tilde{w}$ on $\tilde{L}(P)$, obtained by setting, for $A, B \in \tilde{L}(P)$ (where $A, B$ are population labels),

$$\tilde{w}(A, B) = \min_{\substack{v \in L(P), \phi(v) = A, \\ v' \in L(P), \phi(v') = B}} w(v, v'). \tag{4.1}$$

Note that in our method (that we describe later in the next sections in more details), we do not need to know explicitly either the set $\tilde{P}$ or the surjective map $\phi$. It is reasonable to assume that $w$ and $\tilde{w}$ defined as above satisfy the following properties.

There exists $c > 0$, with $c \ll \mathrm{depth}(P)$, where $\mathrm{depth}(P) = \max_{v \in P} \mathrm{depth}(v)$, and such that

**Property 4.** *For each pair each pair $u, v \in L(P)$,*

*(a) $\phi(u) = \phi(v)$ implies that $w((u, v)) < c$;*

*(b) $\phi(u) \neq \phi(v)$ implies that $w((u, v)) > 2c$;*

*(c) For all $u', v' \in$ with $\phi(u) = \phi(u'), \phi(v) = \phi(v')$, $|w(u, v) - w(u', v')| < c$.*

In other word, Property 4 implies that the distance between two leaf nodes of $P$ carrying the same population label is very small, while those carrying different labels is large, and the latter distance depends only slightly on the chosen representatives, $u, v$, of the respective populations. Property 4 is an ideal property which if satisfied by the data implies a topological result relating the induced Vietoris-Rips complexes on $L(P)$, and on the set of populations $\tilde{L}(P)$ (using the distance measures $w$ and $\tilde{w}$) by virtue of Theorem 4.3 below. Normally, the data will not satisfy this ideal property exactly – but never-the-less we observe a behavior which is close to what the mathematical theorem suggests.

Before stating the precise topological theorem we first explain the main idea.

### 4.3.1 The topological framework

Suppose that in a given finite metric space $M = (V, w)$, where $w : V \times V \to \mathbb{R}_{\geqslant 0}$, the values of $w$ (i.e. the distances) occur in two scales. Suppose also that the points of $V$ form clusters with pairwise distances amongst pairs in each individual cluster belong to the smaller of the two scales – while, the distance between two clusters, measured by taking the minimum of the pairwise distances between the points of the two clusters, belong to the larger scale. We denote the set of clusters by $\tilde{V}$ and the induced metric on $\tilde{V}$ by $\tilde{w}$.

Given any $d > 0$ (where $d$ is "time", in generations, in $P$), the Vietoris-Rips complex of $M$ with parameter $d$, which we denote by $\mathbf{Rips}(M, d)$ (see Definition 4.1), is a certain *simplicial complex* on $V$ (i.e. a family of subsets of $V$ closed under inclusion), and this complex grows with $d$. For small values of $d$ (i.e. closer to the smaller scale) the Vietoris-Rips complex can have complicated topology (measured by the dimensions of the homology groups or the Betti numbers of the complex $\mathbf{Rips}(M, d)$) which depend only on the induced metric spaces on each of the separate clusters. As $d$ grows, the various Vietoris-Rips sub-complexes corresponding to each cluster become contractible, and all homology groups in dimensions $> 0$ vanish (and thus the higher Betti numbers which are the dimensions of these homology groups vanish). After the value of $d$ grows even further (i.e. reaches the larger scale), new homology classes in dimensions $> 0$ might be born and these classes correspond to those of the Vietoris-Rips complex associated to the space $\tilde{M} = (\tilde{V}, \tilde{w})$ obtained from $M$ by clustering.

### 4.3.2 Persistent homology

A systematic way of understanding the birth and death of homology cycles in the Vietoris-Rips complex is through the persistent homology groups [37] (see Definition 4.2 for precise definition). Denoting by $\mathbf{Rips}(M, d)$ the Vietoris-Rips complex of $M$ at "time" $d$, and for all $d' > d$, the inclusion homomorphism $i^{d,d'} : \mathbf{Rips}(M, d) \hookrightarrow \mathbf{Rips}(M, d')$ (which includes $\mathbf{Rips}(M, d)$ in the larger complex $\mathbf{Rips}(M, d')$) induces a homomorphism

$$i_*^{d,d'} : \mathrm{H}_*(\mathbf{Rips}(M, d)) \to \mathrm{H}_*(\mathbf{Rips}(M, d'))$$

between their respective homology groups. Unlike, the homomorphism $i^{d,d'}$, $i_*^{d,d'}$ is not necessarily injective. A non-zero homology class in $\mathrm{H}_*(\mathbf{Rips}(M, d))$ can map to 0 under $i_*^{d,d'}$. The image of $i_*^{d,d'}$ – whose non-zero elements correspond to non-zero homology classes of $\mathrm{H}_*(\mathbf{Rips}(M, d))$ that *persists* till time $d'$, is called the $(d, d')$*-th persistent homology group*, which we will denote by $\mathrm{H}_*^{d,d'}(M)$.

One would expect that the the persistent homology groups of the Vietoris-Rips complex associated to $M$ (in dimensions $> 0$) will also show a separation with respect to the two scales. (The zero-th homology groups will not show such a separation for obvious reasons – and in fact by definition of the Vietoris-Rips complex the zero-th Betti number is just a decreasing function of $d$.) Moreover, one would expect that the homology classes of the Vietoris-Rips complex associated to $\tilde{M}$ which persists over long periods (which are the ones identified with the larger scale) already appear in the persistent homology of the Vietoris-Rips complex associated to $\tilde{M}$, while those associated to the smaller scale appear much earlier and die earlier.

Theorem 4.3 assures us that provided $M, \tilde{M}$ satisfy certain conditions (Property 4) any non-zero persistent homology class in $\mathrm{H}_i^{d,d'}(\tilde{M})$, is the image of a class in $\mathrm{H}_i^{d+c,d'}(M)$ (where $c$ is a constant appearing in Property 4) and can be interpreted as an upper bound on the distances of the smaller scale. Thus, even though we do not have direct access to the Vietoris-Rips complexes of $\tilde{M}$, we can obtain information about its persistent homology from those of the Vietoris-Rips complexes of $M$. In addition, Theorem 4.3 also assures us of the separation on the time scale, of the homology in the Vietoris-Rips complex of $M$ in the smaller time scale, from the "interesting" homology in the larger time scale which contributes to the homology of the Vietoris-Rips complex of $\tilde{M}$. Together they imply that the persistent homology of the Vietoris-Rips complexes of $M$ contains information allowing us to read the persistent homology of the Vietoris-Rips complex $\tilde{M}$ if the latter is non-zero.

### Precise definitions and statement of the topological theorem

To state the topological result alluded to above we first need some definition and notation. We first recall the well known definition of the Vietoris-Rips complex of a finite set $V$ equipped with a distance function $w : V \times V \to \mathbb{R}_{\geqslant 0}$, satisfying $w(v, v) = 0$ for all $v \in V$.

**Definition 4.1** (Vietoris-Rips Complex)**.** Let $M = (V, w)$ be a pair, where $V$ is a finite set and $w : V \times V \to \mathbb{R}_{\geqslant 0}$ is a map (which need not be a metric on $V$) satisfying $w(v, v) = 0$ for all $v \in V$. Then, for any integer $d > 0$, we define the chain complex of the Vietoris-Rips complex of $(M, d)$, which we will denote by $\mathbf{Rips}_\bullet(M, d) = (\mathbf{C}_\bullet(M, d), \partial_\bullet)$ as follows. Let, $V = \{1, \ldots, n\}$, and for each $p \geqslant 0$, define

$$\mathbf{C}_p(M, d) = \bigoplus_{\substack{U \subset V, \\ \mathrm{card}(U)=p+1, \\ \bigwedge_{u,u' \in U} w(u,u') \leqslant d}} \mathbb{Q} \cdot U.$$

The boundary map $\partial_p$ is defined by setting for each $U = \{i_0, \ldots, i_p\} \subset V$, with $1 \leqslant i_0 < \cdots < i_p \leqslant n$, where $U_j = U \setminus \{i_j\}$:

$$\partial_p(U) = \sum_{j=0}^{p} (-1)^j \cdot U_j.$$

**Definition 4.2** (Persistent homology groups of $M$)**.** For $d \leqslant d'$, the inclusion map $i^{d,d'} : \mathbf{Rips}(M, d) \hookrightarrow \mathbf{Rips}(M, d')$ induces homomorphisms $i_\bullet^{d,d'} : \mathbf{Rips}_\bullet(M, d) \to \mathbf{Rips}_\bullet(M, d')$ between the corresponding chain complexes, which in turn induces homomorphisms $i_*^{d,d'} : \mathrm{H}_*(\mathbf{Rips}_\bullet(M, d)) \to \mathrm{H}_*(\mathbf{Rips}_\bullet(M, d'))$ in homology. We call the image of $i_*^{d,d'}$, the $(d, d')$-th *persistent homology group* of $M$ (see for example [37]), and we will denote this group by $\mathrm{H}_*^{d,d'}(M)$.

In [3] the following theorem, which relates the persistent homology groups of two pairs $M = (V, w)$ and $\tilde{M} = (\tilde{V}, \tilde{w})$ under certain conditions, has been proved.

**Theorem 4.3.** *Let $M = (V, w), \tilde{M} = (\tilde{V}, \tilde{w})$ be as in above with $V, V'$ finite, $c > 0$ and $\phi : V \to V'$ a surjective map, such that for each pair $u, v \in V$ satisfies Property 4. Then,*

1. $\mathrm{H}_i(\mathbf{Rips}_\bullet(\tilde{M}, d)) = 0$ *for* $i > 0$, *and* $d < 2c$.
2. *For all $d, d' \geqslant 0$ satisfying $d' - d > 2c$ $\phi$ induces a surjective homomorphism*

$$\phi_*^{d,d'} : \mathrm{H}_*^{d+c,d'}(M) \longrightarrow \mathrm{H}_*^{d,d'}(\tilde{M}).$$

*Moreover, if $\tilde{i}_*^{d,d'} : \mathrm{H}_*(\mathbf{Rips}_\bullet(\tilde{M}, d)) \longrightarrow \mathrm{H}_*(\mathbf{Rips}_\bullet(\tilde{M}, d'))$ is an isomorphism, then so is $\phi_*^{d,d'}$.*

We report here the proof of the theorem from [3]. *Proof*: The first claim immediately follows from Part (c) of Property 4. We now prove the second claim. We first check that for any $d > 0$, the map $\phi$ induces a simplicial map $\phi : \mathbf{Rips}(M, d) \to \mathbf{Rips}(\tilde{M}, d)$. To see this let $U \subset V$ such that $\bigwedge_{u,u' \in U} w(u, u') \leqslant d$. We claim that for each $u, u' \in U$, $\tilde{w}(\phi(u), \phi(u')) \leqslant d$. This follows immediately from the definition of $\tilde{w}$ (see Eqn 4.1). Notice that the min function used in the definition of $\tilde{w}$ is crucial here. This proves that the induced map of $\phi$ is simplicial i.e. it carries simplices to simplices. Now suppose that $d' - d > 2c$, and consider a simplex in the Vietoris-Rips complex $\mathbf{Rips}(\tilde{M}, d)$ spanned by $\tilde{U} \subset \tilde{V}$. Since, $\tilde{U}$ is a simplex in the Vietoris-Rips complex, $\mathbf{Rips}(\tilde{M}, d)$, by definition $\bigwedge_{\tilde{u},\tilde{u}' \in \tilde{U}} \tilde{w}(\tilde{u}, \tilde{u}') \leqslant d$. Then, for all $u \in \phi^{-1}(\tilde{u}), u' \in \phi^{-1}(\tilde{u}')$, $w(u, u') \leqslant d + c$, (using Parts (a) and (b) of Property 4 ). Thus, the inverse image of the simplex spanned by $\tilde{U}$ in $\mathbf{Rips}(\tilde{M}, d)$, is contractible inside $\mathbf{Rips}(M, d + c) \hookrightarrow \mathbf{Rips}(M, d')$. It now follows by an application of the Vietoris-Begle theorem (see for example [38, page 344]), that the

induced homomorphism $\phi_*^{d,d'} : \mathrm{H}_*^{d+c,d'}(M) \to \mathrm{H}_*^{d,d'}(\tilde{M})$ is a surjective homomorphism, and is an isomorphism if $\tilde{i}_*^{d,d'} : \mathrm{H}_*(\mathbf{Rips}_\bullet(\tilde{M}, d)) \to \mathrm{H}_*(\mathbf{Rips}_\bullet(\tilde{M}, d'))$ is an isomorphism. □

Theorem 4.3 is applicable in the context of Scenario III as follows. Take $M = (L(P), w)$ and $\tilde{M} = (\tilde{L}(P), \tilde{w})$. Further suppose that for $A, B \in \tilde{L}(P), A \neq B, |\tilde{w}(A) - \tilde{w}(B)| > 2c$ (say), that is distinct populations are separated by a larger distance than individuals within the same population.

In this case the surjection given in Theorem 4.3 implies that the presence of persistent homology (i.e. homology cycles that are born after $d = 2c$ and that persists for intervals of length $> c$) in the Vietoris-Rips complex of $\tilde{M}$ can be detected from that of the Vietoris-Rips complex of $M$. Hence, for all small values of $d, d'$, i.e. $0 < d < d' < c$, the persistent homology groups, $\mathrm{H}_*^{d,d'}(M)$ reflect the topology of the ARG $P$, created by the recombination events. For $c < d < d + 2c < d'$ by Theorem 4.3, there is a surjection

$$\mathrm{H}_*^{d+c,d'}(M) \to \mathrm{H}_*^{d,d'}(\tilde{M}).$$

which is an isomorphism if $\tilde{i}_*^{d,d'}$ is an isomorphism, and any persistent homology (in dimension $> 0$) in this range can be attributed to the cycles in the population graph $P'$ which are caused by admixture.

### 4.3.3 Topological Signatures

The theorem thus predicts that the presence of admixtures should be detectable from the persistent homology diagrams of the Vietoris-Rips complex of $M$ itself. This is indeed seen in the experimental results. In Figs 4.3-4.6, we display the results of computing the homology groups of the Rips complexes obtained from both simulated as well as real data. We take $M = (L(P), w)$ where $P$ is an ARG obtained either from simulated or real data. Fig 4.7 shows results for real data while the others are for simulated data. The horizontal axis corresponds to the values of $d$, and for each fixed $d$, the number of horizontal lines above is the dimension of homology group of the Vietoris-Rips complex corresponding to this value of $d$. Thus, each horizontal line depicts the "life" of a non-zero homology cycle. The $x$-coordinate of its left end point is the time of its "birth" and the right end point the time of its "death". We see a clear separation between persistent cycles in dimensions $> 1$, in the case of admixed populations – which can be seen as a signal indicating presence of admixture.

## 4.4 Experiments on simulated data

We first describe the simulation experiments. The populations were simulated using SimRA (See Chapter 3). Once the set of haplotypes were generated for all three populations, we created a distance matrix between all pairs of haplotypes using Hamming distance metric. The Vietoris-Rips complex was constructed on the graph embedding of the distance matrix (a complete graph with each vertex corresponding to an individual haplotype and edge weights corresponding to the Hamming distance between the pair of haplotypes). We computed homology groups on the Vietoris-Rips complex for zero and one dimensions using Javaplex v4.2.0 [39]. Recall that the dimension of the zero-dimensional homology group of a simplicial complex counts the number of connected components of the simplicial complex, while the dimension of the one-dimensional homology group counts the number of independent one-dimensional cycles which do not bound.

In the results, irreducible cycles computed from the simulation experiments are presented as barcode plots, which display when individual cycles representing non-zero one-dimensional homology classes are born and when they disappear. The upper half of each barcode plot for the simulation experiments display the persistence of zero-dimensional homology, while the lower half display barcode line segments indicate the persistence of one-dimensional homology. While short cycles can be due to noise, longer (persistent) cycles represent fundamental topological structures within the genetic distance matrix.

Fig 4.3 shows the topological signatures in the context of presence and absence of admixture. The persistent cycles for dimension $> 0$ clearly separate into two groups. Figs 4.5-4.6 show the results of experiments with different simulation parameters, including stochasticity of the ARGs.

## 4.5 Experiments on avocado germplasm

We consider three main avocado cultivars: West Indian (W), Guatamelan (G) and Mexican (M). Moreover, we also consider an $F_1$ population WxG. Each of the group is composed of 19 samples, from which we have 3348 markers. The genotype data were phased using Beagle [40] and both haplotypes are used in our experiments. In particular, using these four groups, we created two datasets to match our simulation study set-up: one composed of W, G and WxG samples and the other of G, M and W. The former set admits admixture while the latter does not.

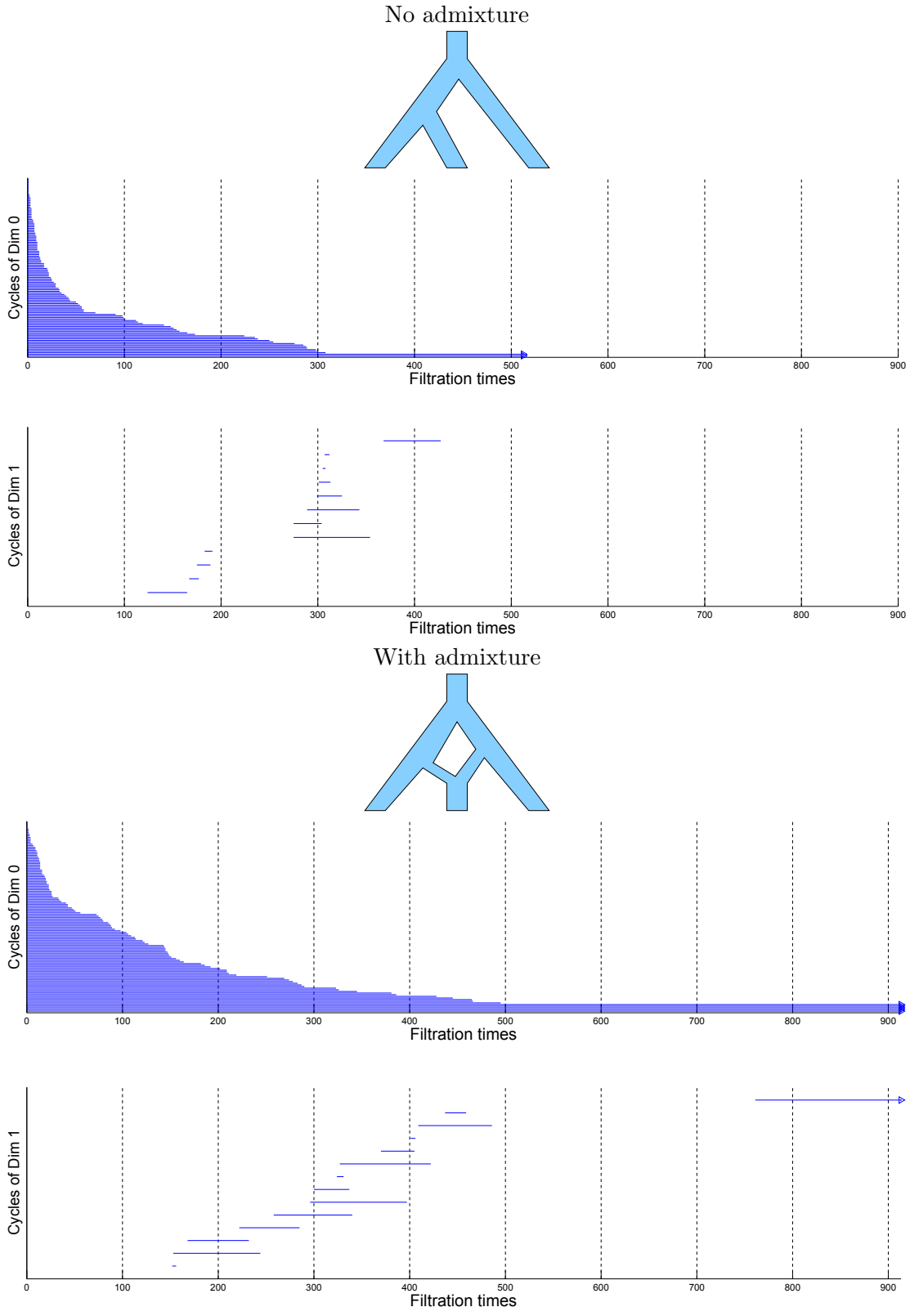



FIGURE 4.3: Topology signatures embedded in the ARGs, on simulated data. There is an absence of admixture in the top while a presence in the bottom panel. This proof-of-concept experimental setting shows that, in ideal scenarios of simulations, topological signatures for recombinations and admixture can be differentiated (notice, in particular, the separation of the persistent cycles of dimension $> 0$). In the simulations, the effective population size is $N = 10K$. See text for further details.

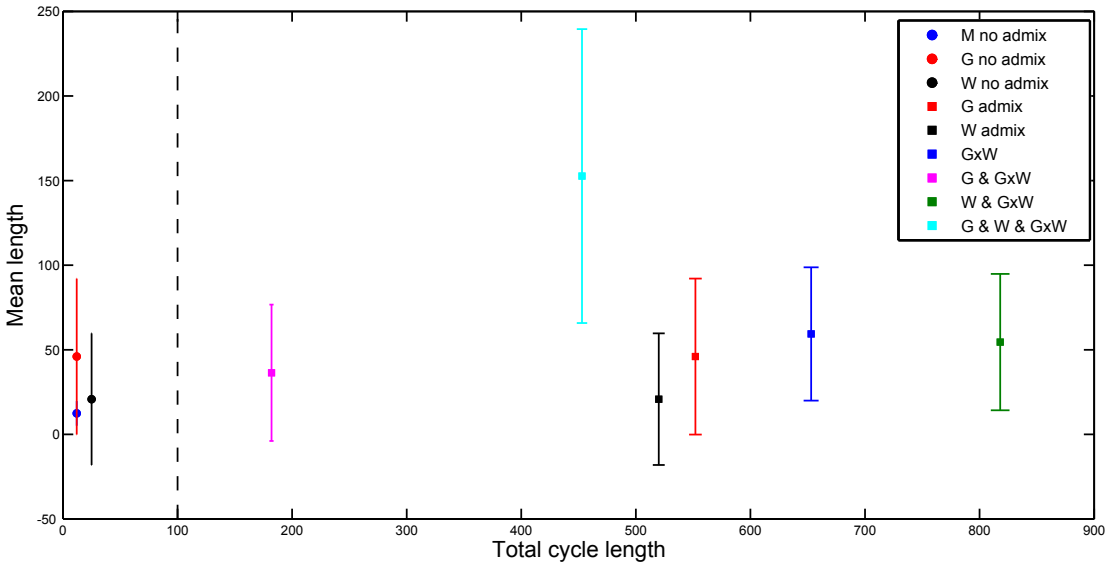



FIGURE 4.4: Analysis of the persistent cycles of avocado germplasm data: It shows that the admixed samples have larger cycle lengths ($> 100$). The mean length for the 3 admixed populations is larger than the other cases. Also, the individuals for the G and W cultivars are the same in both the experiments, and they have comparable mean length (the red and black lines in the plot), while the total cycle lengths are different.

In order to compute the persistent homology groups on the avocado germplasm data, we concatenated SNP loci from all 12 chromosomes into a single sequence for each haplotype and computed the distance matrix based on the Hamming distance metric as described above. For the two avocado germplasm datasets, we computed zero, one and two-dimensional cycles representing non-zero elements of the persistent homology groups on the Vietoris-Rips complex using Javaplex. Fig 4.7 shows barcode plots describing zero, one and two-dimensional topological signatures on these two avocado germplasm data sets with and without admixture present. Further analysis of the persistent cycles in terms of their mean length and variances again shows distinguishing characteristics: see Fig 4.4.

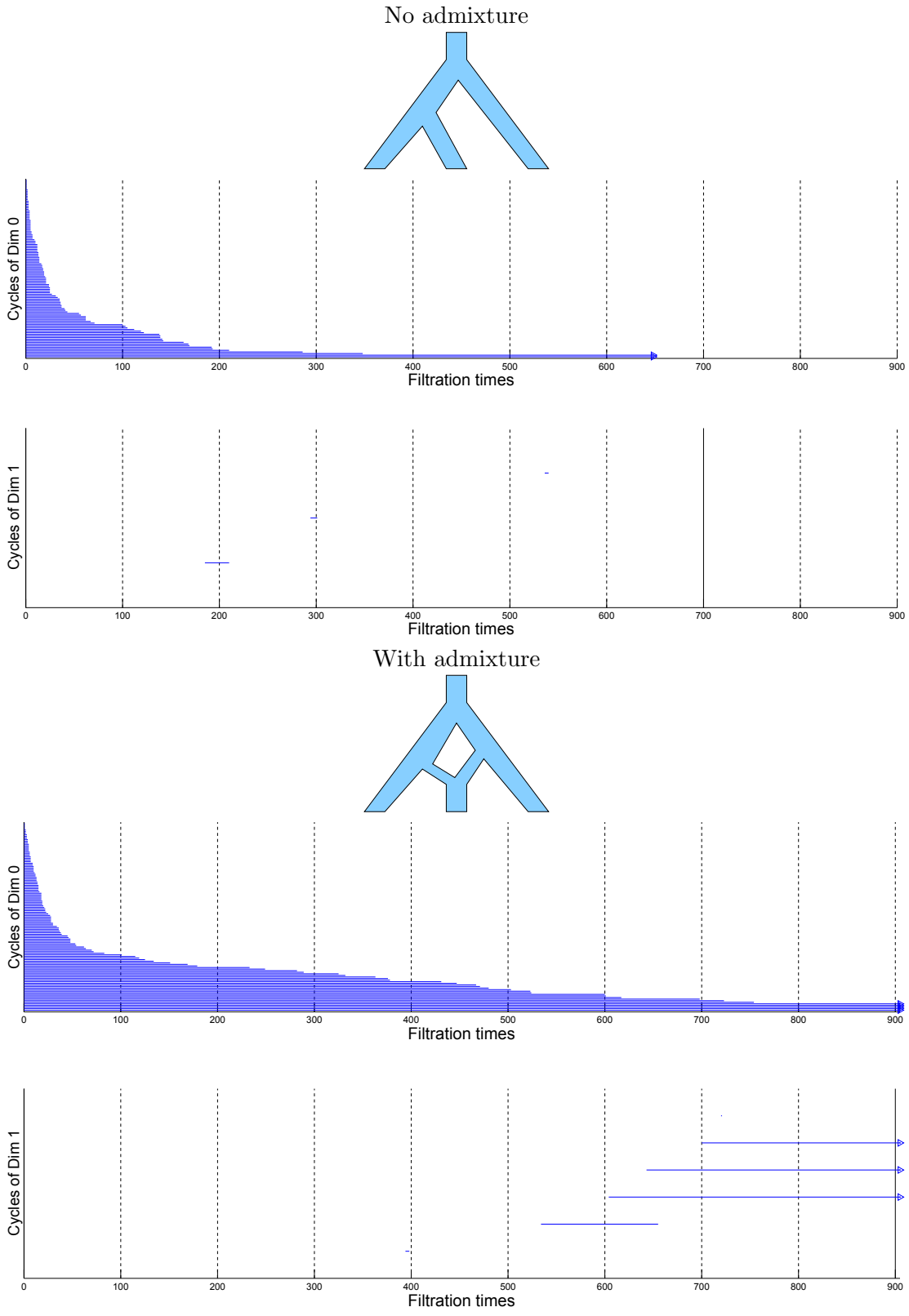



FIGURE 4.5: In the simulations recombination rate $r = 0.1 \times 10^{-8}$. Notice that in the absence of recombinations, no particular separation of persistent cycles is observed. In the simulations, the effective population size is $N = 10K$.

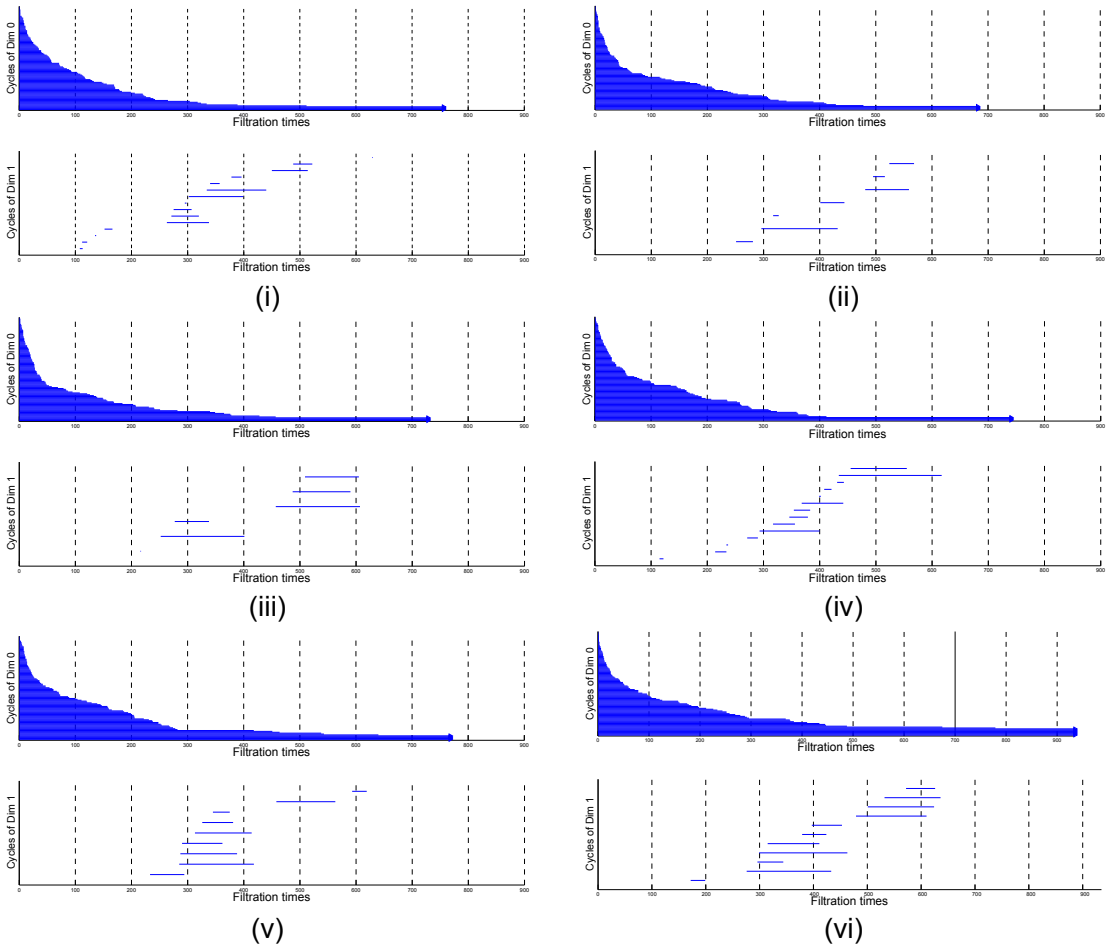



FIGURE 4.6: Six simulations, each with effective population size $N = 10K$; with recombination ($r = 0.3 \times 10^{-8}$) as well as admixture to show that stochasticity does not affect the topological signature, i.e. the separation of the persistent cycles into roughly two groups.

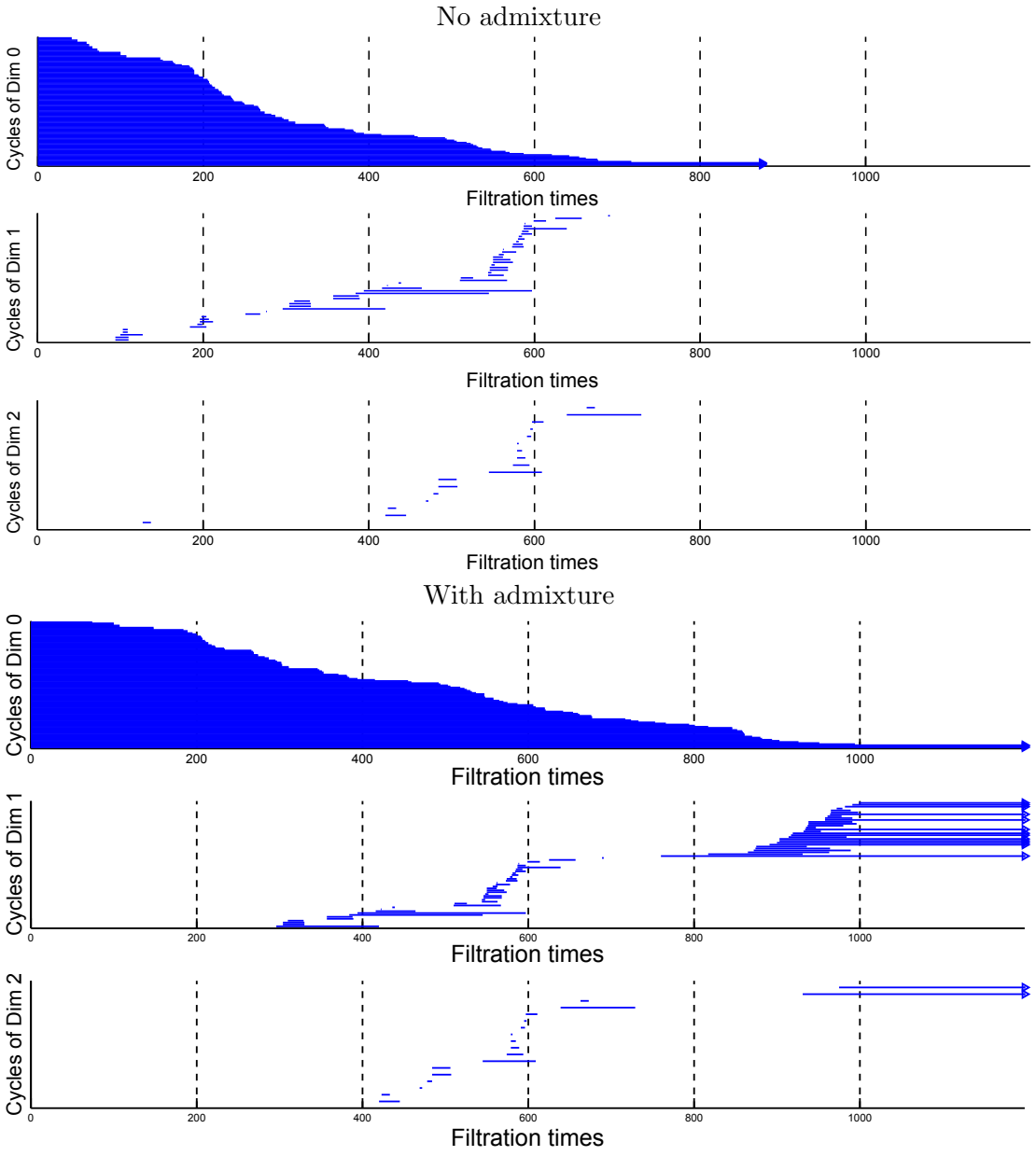



FIGURE 4.7: Haplotypes from three groups of avocado germplasm data: West Indian (W), Mexican (M), and Guatemalan (G). The top plot corresponds to the populations with no admixture, while the bottom admits admixture in the populations. Notice the separation of the persistent cycles in both dimension 1 and 2 for the latter scenario, while the former shows no clear separation.

# Chapter 5

# Character based methods to reconstruct phylogenetic trees

In this chapter we introduce the state of art regarding computational model and methods for phylogenetic reconstruction. This is useful for a better understanding of the results obtained while investigating the second research direction, that is the development of combinatorial algorithms for the reconstruction of trees representing the evolutionary history of a set of species. In particular, we present a survey of character based methods to reconstruct phylogenesis, mainly following the line of [4], and we focus our attention on the perfect phylogeny model and one of its variant, called persistent perfect phylogeny.

## 5.1 Motivation

Phylogenetics is the research area of Computational Biology devoted to computing phylogenies. A *phylogeny* is a prototypical representation of any evolutionary history, that is a labeled tree whose leaves are the extant species, or individuals, or simply biological data, such as genomic sequences, that we are currently able to analyze [41].

In this field, the focus has shifted through the years. The initial developments date back to the pioneering work by Cavalli-Sforza and Edwards [42] in the 60s, where some fundamental ideas of the study of phylogeniesis have been introduced, namely the fact that evolution is a branching process where characters or attributes of species are changing.

Initially the emphasis has be focused on character-based approaches due to the limited computational resources of time, together with the kind of data available (phenotypical data (characters) were much more frequent than genomic data). Character-based methods are based on maximum parsimony models that want to minimize the total amount of

evolutionary events compatible with available data. Successively new advances, also in the statistical modeling of evolution [41], made the approaches based on inferring maximum likelihood phylogenies more attractive. More recently, the pendulum has swung again, as parsimony models and character-based methods have found new relevance, mostly due to new applications and available data. Recent applications, indeed, show that parsimony models can be applied to analyze the evolution of data related to various genomic information, such as protein domains [43] and markers in tumor [44].

## 5.2 Maximum Parsimony Models and the perfect phylogeny

Maximum parsimony models are based on specific constraints deriving from some biological assumptions. The first basic assumption states that each species or taxon is described by a set of attributes, called characters, where each character is inherited independently and can assume one of a finite set of values, also called *states*. Another basic assumption about the evolution of characters, called *homology*, assumes that characters that are present in more than one species must be inherited by a common ancestor. In other words, parsimony models assume a coalescent model, i.e., a characteristic shared by a set of species can be traced back to a single ancestral species.

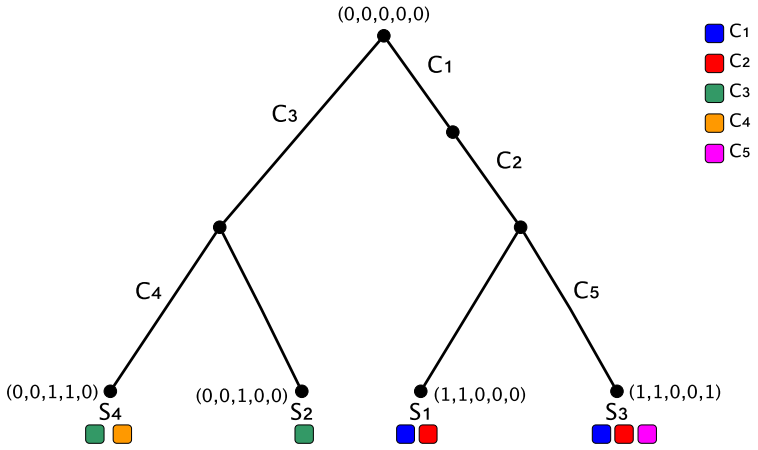
The natural computational problem has, as input, a matrix $M$ with $n$ rows and $m$ columns, where each row can be viewed as a $m$-vector over the set of character states. The matrix describes a set of $n$ taxa (species or individuals) – corresponding to the rows of $M$ – and a set of $m$ characters – corresponding to the columns of $M$ – and asks for a minimum cost tree explaining the input matrix $M$. In a tree $T$ *explaining* a matrix $M$ (i) the nodes are labeled by $m$-long vectors of character states, (ii) each row of $M$ labels exactly one node of $T$, (iii) the leaves are labeled by some rows of $M$ (iv) each edge $(r_1, r_2)$ of $T$ is labeled by the character $c$ of $M$ whose state differs in $r_1$ from that in $r_2$ (see Figure 5.1 as an example). The cost of a tree is the number of state changes or transitions of characters, called *mutations*.

We consider binary parsimony models – the most widely used – where characters can take only the values (or states) zero or one, usually interpreted as the presence or absence of an attribute in the taxa or species.

We can now introduce some specific parsimony models, starting from the simplest and most restrictive: the *perfect phylogeny*. This parsimony model, that is one of the most investigated [45], is conceptually based on the *infinite sites assumption*, i.e., no character can mutate more than once in the whole tree. More precisely, in a perfect phylogenetic tree each character $i$ mutates exactly once (i.e., there is exactly one edge such that the

| $M$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ |
|---|---|---|---|---|---|
| $s_1$ | 1 | 1 | 0 | 0 | 0 |
| $s_2$ | 0 | 0 | 1 | 0 | 0 |
| $s_3$ | 1 | 1 | 0 | 0 | 1 |
| $s_4$ | 0 | 0 | 1 | 1 | 0 |

(a) Binary matrix $M$




(b) A perfect phylogeny $T$

FIGURE 5.1: Example of perfect phylogeny $T$ in Fig. 5.1(b) for a binary matrix $M$ in Fig. 5.1(a) of 5 characters and 4 species. In Fig. 5.1(b) each character has associated a different color. The leaves of $T$ are labeled with the species of $M$ and the colors of characters they own.

vertices are labeled by vectors differing in position $i$). Notice that a perfect phylogeny (if it exists) minimizes the overall cost, as any perfect phylogeny has cost $m$ corresponding to the number of characters in the input binary matrix $M$.

We call the perfect phylogeny *directed* or *rooted* if there is a distinguished node (root of the tree) corresponding to the $[0, \ldots, 0]$ vector. We focus on the rooted binary perfect phylogeny and we give a formal definition from [41].

**Definition 5.1** (perfect phylogeny)**.** A rooted binary tree $T$ explaining a binary matrix $M$ is called perfect phylogeny if and only if the following holds:

1. the root is labeled with a vector of all zero $[0 \ldots 0]$;
2. each character $c$ of $M$ univocally labels an edge in $T$ representing the state transition (or mutation) from 0 to 1 of the character $c$;
3. characters that label edges on the path from the root to a leaf $i$ correspond to the characters that have state 1 for the species $i$.

An example of perfect phylogeny is in Fig. 5.1.

It is immediate that we can transform an un-rooted perfect phylogeny into a rooted perfect phylogeny, by choosing an arbitrary node $x$ and flipping (in each species) the state of each character that initially has value 1 in $x$ (those characters are also called *active* in $x$). However in the following, unless specified differently, by perfect phylogeny we mean rooted perfect phylogeny.

The binary perfect phylogeny reconstruction problem has received much attention. A linear time algorithm when all data is known [46] and a near-optimal time algorithm when the input data is incomplete [47] have been presented in literature.

The perfect phylogeny model turned out to be coherent and widely applied within the haplotyping problem [48, 49], where we want to distinguish the two haplotypes present in each individual when only genotype data is given. More precisely, the interest here is in computing a set of haplotypes and a perfect phylogeny such that the haplotypes (i) label the vertices of the perfect phylogeny and (ii) explain the input set of genotypes. This context has been deeply studied in the last decade, giving rise to a number of algorithms [50, 51].

Even though the perfect phylogeny has been central in the previous decades, the infinite state assumption is too restrictive for explaining many other kinds of real biological data and cannot be applied without adaptations or improvements. A first generalization allows characters to assume more states (but keeping the infinite sites assumption). In the general case, the problem is NP-hard [52], but fixed-parameter tractable algorithms for the general perfect phylogeny problem, where the parameter is the maximum number of states for each character, have been proposed [53, 54]. The special cases when there are three or four possible states have more efficient algorithms [55–57].

Even allowing more states for each character, the perfect phylogeny still cannot explain the biological complexity of real data. Unfortunately there are some evolutionary phenomena, such as *homoplasies*, that violate the fundamental assumptions of perfect phylogeny [41]. Two kinds of homoplasies are *recurrent mutations* and *back mutations*. The first event occurs when a character changes state along divergent branches of the tree, while a back mutation implies that a character may go back to the ancestral state in descendant species after changing its state. Two cases where the limitation of the perfect phylogeny in presence of homoplasies is evident are the study of *carcinogenesis* and *protein domains evolution*.

A protein domain is a part of protein sequence and structure that can evolve independently of the rest of the protein chain. Many proteins consist of several structural domains, while a domain may appear in a variety of different proteins. In this case it is quite frequent to acquire a domain and then lose it [43].

Carcinogenesis consists of the factors and mechanisms that cause the onset of cancer; it results from many combinations of mutations, but only a few, called progression pathways, seem to account for most human tumors [58]. The observation that tumors are evolving cell populations leads to phylogeny-based studies. At the same time, the intrinsic nature of quickly and degenerately proliferating cancer cells, results in a relative high amount of sites with multiple mutations (i.e., in violations of the infinite sites assumption). The studies of carcinogenesis reveal that data do not always fit the perfect phylogeny model, since some mutations lead to inconsistency in the data that must be solved by their removal. Detection of inconsistencies that have to be removed is a crucial task of many tools for single nucleotide variants (SNVs) calling in tumor tissues [59]. Thus relaxing the infinite site assumption, allowing for example back mutations, may solve this issue.

While the perfect phylogeny model does not model any homoplasy, some extended parsimony models have been introduced to allow recurrent or back mutations. In the following we present some established binary parsimony models that mainly differ from different kinds of restriction on the type of state transitions from zero to one and vice versa [41].

## 5.3 Camin Sokal Parsimony, Dollo Parsimony and its variants

A first extended model is the Camin-Sokal parsimony [60], where characters are *directed*, i.e., only changes from zero to one are possible on any path from the root to one leaf of a tree $T$ . This fact means that the root of $T$ is assumed to be labeled by the ancestral state with all zeroes, and no back mutation is allowed, but any character can be acquired more than once, i.e., recurrent mutations are possible. Fig. 5.2 gives an example of Camin-Sokal parsimony tree.

Another maximum parsimony model that relaxes the strict assumption of the perfect phylogeny model is the Dollo Parsimony. The latter model requires that characters can be acquired at most once in the tree, but may be lost multiple times. In other words, the Dollo parsimony allows any character to change state from zero to one only once, but gives no restriction on the number of times that each character can mutate from one to zero [41]. See Fig. 5.3 for an example.

Notice that, differently from the perfect phylogeny, a rooted Dollo phylogeny always exists for an input matrix $M$. Indeed if we assume a special internal node $[1, \ldots, 1]$, that is also the least common ancestor of all leaves, any binary vector can be generated, as there is no restriction on the number of mutations from 1 to 0. Even though there

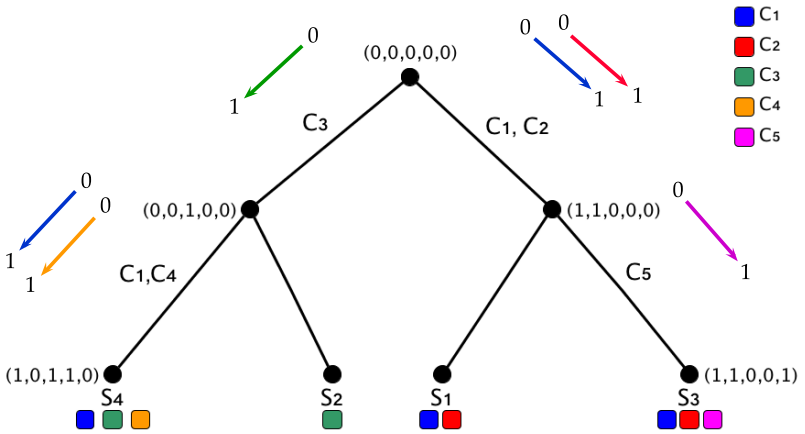



FIGURE 5.2: Example of Camin-Sokal parsimony. Each character has a different color. The edges are labeled with colored arrows indicating state transitions from 0 to 1 of the corresponding characters. Observe that $c_1$ (blue arrows) is gained twice in the tree.

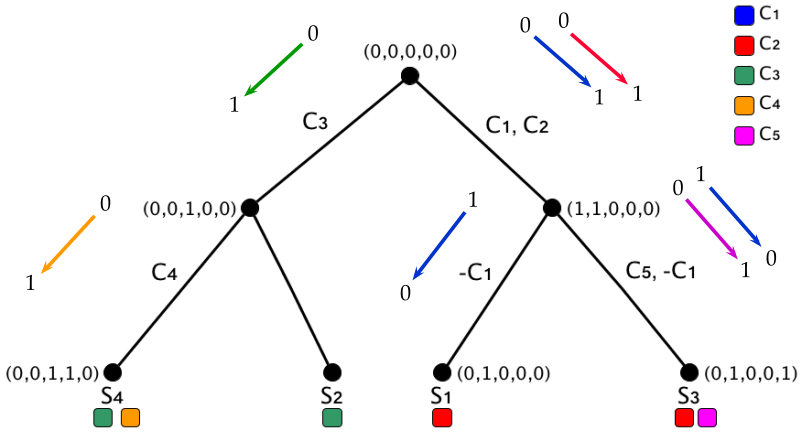



FIGURE 5.3: Example of Dollo parsimony over a matrix of five characters. Each character has a different color. The edges are labeled with colored arrows indicating state transitions from 0 to 1 or from 1 to 0 of the corresponding characters. Observe that $c_1$ (blue arrows) is the only one that is lost twice on two different paths of the tree.

is no guarantee that such tree is optimal, it suffices to prove the existence of a Dollo phylogeny. However such a tree does not make sense from a biological point of view, because it implies the existence of an ancestral taxon that has all characters that are present in the extant taxa.

An interesting application of the Dollo parsimony is the analysis of dynamic protein interactions [61], which has also shown an interesting connection with graph theory. Protein networks are graphs modeling protein interactions. More precisely, nodes are proteins under analysis and edges represent the interactions among proteins.

As pointed out above, the perfect phylogeny model is too restrictive for some applications, since it cannot explain the evolution of characters in the presence of homoplasy events.

On the other hand, the optimization problems associated to the Dollo and Camin-Sokal parsimony models are NP-hard [41]. Moreover, these models are often too general to be useful in practical applications where characters are usually affected only by a few back mutations or recurrent mutations. Therefore the research activity has been focused on developing new models that relax the strict assumption of the perfect phylogeny in order to adequately model actual phenomena and maintain the computational efficiency at the same time. For example, in the context of proteomics when analyzing properties of multi-domain proteins [43, 61].

The problem of constructing phylogenies with deviations from perfect phylogeny has been tackled under the name of near perfect phylogeny [62] or near perfect phylogeny haplotyping problems [63]. Especially the impossibility of losing a character that has been previously acquired is too restrictive, resulting in more elaborated models, such as the *persistent* character [43] and the General Cladistic Character Compatibility (GCCC) [64, 65].

In particular, the persistent perfect phylogeny model [66] allows each character to be lost (i.e., can chance its state from 1 to 0) in at most one edge of the phylogeny, while the General Cladistic Character Compatibility imposes some restrictions on the possible mutations (i.e., on the possible states labeling the endpoints of an edge), while allowing the input data to be a set of possible states for each character of a species.

In the next section, we present the persistent perfect phylogeny model, the basis of the results presented in Chapter 6.

## 5.4 The persistent perfect phylogeny

An important ingredient that may affect the applicability and success of parsimony models is given by the set of characters used to infer the phylogeny. The issue of selecting characters has been addressed in [61], where the notion of *persistent* or *stable* character has been proposed. Such characters are allowed to violate the properties of a perfect phylogeny, as a persistent character is gained and lost exactly once in the whole tree.

Based on this notion, a different model, which is a generalization of perfect phylogeny, called the *persistent perfect phylogeny* has been proposed in [66]. Note that a persistent perfect phylogeny is also a Dollo phylogeny, but differently from Dollo Parsimony, some binary matrices may not admit a persistent perfect phylogeny.

Therefore, the main computational problem we discuss in this section is to compute (if it exists) a persistent perfect phylogeny describing the evolutionary history of a given

matrix $M$. We notice that the computational complexity of this problem is still unsettled, while there exists an algorithm that is exponential in the number of characters, but polynomial in the number of species [66]. This time complexity makes the algorithm of practical interest for biological applications we have discussed before, as usually the number of species is large, while the number of characters is bounded.

More precisely, the problem is called Persistent Perfect Phylogeny (PPP) reconstruction problem and it is formally defined as follows.

### 5.4.1 The Persistent Perfect Phylogeny (PPP) Problem

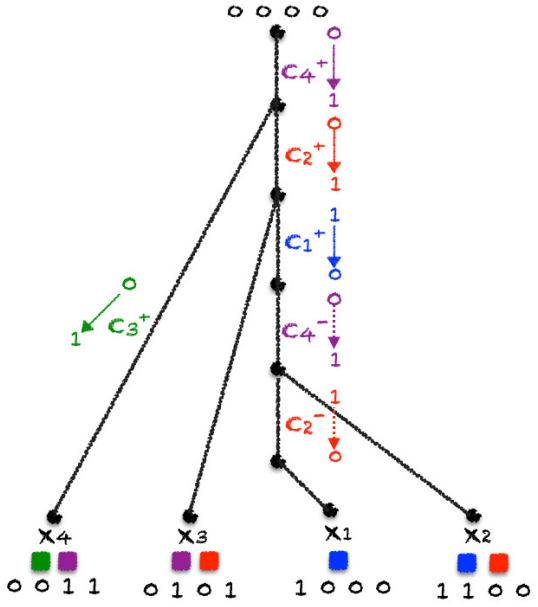
Given a binary matrix $M$ in input, the PPP problem asks for finding a PPP for $M$ if such a tree exists. Hence, the input is an $n \times m$ binary matrix $M$ where columns are associated with the set of characters $C = \{c_1, \ldots, c_m\}$ and rows are associated with the set $S = \{s_1, \ldots, s_n\}$ of species. Then $M[i, j] = 1$ if and only if the species $s_i$ has the character $c_j$, otherwise $M[i, j] = 0$. The character $c$ is *gained* on the only edge where its state goes from 0 to 1 or, more formally, on the edge $(x, y)$ such that $y$ is a child of $x$ and $c$ has state 0 in $x$ and state 1 in $y$. In this case the edge $(x, y)$ is labeled by $c^+$. Conversely, $c$ is *lost* on the edge $(x, y)$ if $y$ is a child of $x$ and $c$ has state 1 in $x$ and state 0 in $y$. In the latter case the edge $(x, y)$ is labeled by $c^-$. For each character $c$, the persistent perfect phylogeny model allows at most one edge labeled by $c^-$.

**Definition 5.2** (persistent perfect phylogeny)**.** Let $M$ be an $n \times m$ binary matrix. Then a persistent perfect phylogeny for $M$ is a rooted tree $T$ such that:

1. each node $x$ of $T$ is labeled by a vector $l_x$ of length $m$;
2. the root of $T$ is labeled by a vector of all zeroes, while for each node $x$ of $T$ the value $l_x[j] \in \{0, 1\}$ represents the state of character $c_j$ in tree $T$;
3. each edge $e = (v, w)$ can be labeled by one character, but only edges incident on a leaf can be unlabeled;
4. for each character $c_j$ there are at most two edges $e = (x, y)$ and $e' = (u, v)$ such that $l_x[j] \neq l_y[j]$ and $l_u[j] \neq l_v[j]$ (representing a state transition of $c_j$). In that case $e$, $e'$ occur along the same path from the root of $T$ to a leaf of $T$; if $e$ is closer to the root than $e'$, then $l_x[j] = l_v[j] = 0$, $l_y[j] = l_u[j] = 1$, and the edge $e$ is labeled $c_j^+$, while $e'$ is labeled $c_j^-$;
5. there is a 1-to-1 correspondence between each row $r$ of $M$ and each leaf $x$ of $T$. Moreover the vector $l_x$ corresponds to row $r$.

| $M$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ |
|---|---|---|---|---|
| $x_1$ | 1 | 0 | 0 | 0 |
| $x_2$ | 1 | 1 | 0 | 0 |
| $x_3$ | 0 | 1 | 0 | 1 |
| $x_4$ | 0 | 0 | 1 | 1 |

(a) Binary matrix $M$




(b) Persistent perfect phylogeny $T$ for $M$

FIGURE 5.4: Example of persistent perfect phylogeny over a binary matrix $M$ of 4 characters and 4 species. The species $x_1, x_2, x_3, x_4$ of $M$ in Fig. 5.4(a) label the leaves in the persistent phylogeny in Fig. 5.4(b). At each character of $M$ we associate a different color and the edges in $T$ are labeled with colored state transitions from 0 to 1 or from 1 to 0. Note that $c_4$ and $c_2$ are persistent characters since they mutate twice.

Then we say that the binary matrix $M$ is solved by tree $T$.

Fig. 5.4 shows an example of persistent phylogeny for an input binary matrix $M$ of size $4 \times 4$.

The requirement that only edges incident on a leaf might be unlabeled is technical and stems from the fact that we want to establish a 1-to-1 correspondence between rows of $M$ and leaves of $T$. It is possible to give an equivalent definition, where each edge is labeled by at least one character (or exactly one character), but in that case we would need to associate also internal nodes to rows of $M$ (see the definition of phylogenetic network [67] for such an example).

We give below a formal definition of persistent character.

**Definition 5.3** (persistent character)**.** Let $s$ be a species and let $c$ be a character such that, in a persistent perfect phylogeny $T$, the path from the root of $T$ to $s$ traverses one edge labeled $c^-$. Then $c$ is called *persistent* for $s$ in $T$.

### 5.4.2 The extended matrix, the red black graph and the conflict graph

We can restate the PPP problem as a variant of the Incomplete Directed Perfect Phylogeny [47] by associating to the input matrix $M$ an incomplete matrix called extended matrix $M_e$ defined as follows.

**Definition 5.4** (extended matrix)**.** Let $M$ be an instance of the PPP problem. The *extended matrix* associated with $M$ is an $n \times 2m$ matrix $M_e$ over alphabet $\{0, 1, ?\}$ which is obtained by replacing each column $c$ of $M$ by a pair of columns $(c^+, c^-)$, where $c^+$ is called the *positive* character, and $c^-$ is called the *negated* character. Moreover for each row $s$ of $M$, $M_e[s, c^+] = 1$ and $M_e[s, c^-] = 0$ if $M[s, c] = 1$, while $M_e[s, c^+] = M_e[s, c^-] =?$ if $M[s, c] = 0$. The symbol ? means that the value of such cell is unknown.

Fig. 5.5 provides an example of extended matrix $M_e$ associated to a matrix $M$.

The goal is to complete the extended matrix $M_e$ obtaining a new matrix $M_f$ by assigning values to each ? entry. The pairs of characters $(c^+, c^-)$ are called *conjugate*. A *completion* of a pair $(?, ?)$ associated to a species $s$ and conjugate characters $(c^+, c^-)$ of $M_e$ consists of forcing $M_e[c^+, s] = M_e[c^-, s] = 0$ or $M_e[c^+, s] = M_e[c^-, s] = 1$.

Informally, the assignment of a conjugate pair $(?, ?)$ in a species row $s$ for two conjugate characters $(c^+, c^-)$ with $(1, 1)$ means that character $c$ is persistent for species $s$, i.e., it is first gained and then lost on the path from the root to the leaf $s$. In the latter case $M_f[s, c^-] = 1$ is interpreted as the fact that the species $s$ does not have the character $c$, but some of its ancestors used to have it. On the contrary, the assignment of $(1, 0)$ means that character $c$ is only gained on the path from the root to the species $s$.

A *partial completion* of the extended matrix $M_e$, associated with an input matrix $M$, is a completion of some of its conjugate pairs, while a completion $M_f$ is *full* if all its conjugate pairs are completed.

**Observation 2.** *The idea of completing a matrix with missing data in order to obtain a perfect phylogeny has been introduced in [47], but in our case the completion has some constrains, making the algorithm of [47] inapplicable. Indeed, the completion of a conjugate pair* $(?, ?)$ *is required to be either* $(0, 0)$ *or* $(1, 1)$.

| $M$ | a | b | c | d | e |
|---|---|---|---|---|---|
| $s_1$ | 0 | 0 | 0 | 1 | 0 |
| $s_2$ | 0 | 0 | 1 | 1 | 1 |
| $s_3$ | 0 | 1 | 1 | 0 | 0 |
| $s_4$ | 1 | 1 | 0 | 0 | 0 |
| $s_5$ | 1 | 1 | 1 | 0 | 1 |

| $M_e$ | $a^+$ | $a^-$ | $b^+$ | $b^-$ | $c^+$ | $c^-$ | $d^+$ | $d^-$ | $e^+$ | $e^-$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $s_1$ | ? | ? | ? | ? | ? | ? | 1 | 0 | ? | ? |
| $s_2$ | ? | ? | ? | ? | 1 | 0 | 1 | 0 | 1 | 0 |
| $s_3$ | ? | ? | 1 | 0 | 1 | 0 | ? | ? | ? | ? |
| $s_4$ | 1 | 0 | 1 | 0 | ? | ? | ? | ? | ? | ? |
| $s_5$ | 1 | 0 | 1 | 0 | 1 | 0 | ? | ? | 1 | 0 |

FIGURE 5.5: An example of binary matrix $M$ (which is the input of the persistent phylogeny problem) and its associated extended matrix $M_e$.

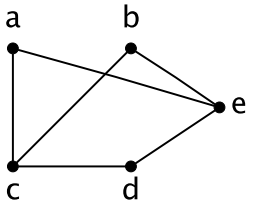



FIGURE 5.6: The conflict graph $G_c$ associated with the binary matrix $M$ of Fig. 5.5.

Finding such a full completion $M_f$ of an extended matrix $M_e$, associated with an input matrix $M$, that admits a perfect phylogeny is equivalent to computing a persistent phylogeny on the original matrix $M$. The following Theorem has been proved in [66].

**Theorem 5.5.** *Let $M$ be a binary matrix and $M_e$ the extended matrix associated with $M$. Then $M$ admits a persistent phylogeny if and only if there exists a completion $M_f$ of $M_e$ admitting a perfect phylogeny.*

We devote the remainder of the section to the discussion of the algorithm of [66] for determining whether an input matrix $M$ admits a persistent perfect phylogeny and, in case, to compute such a phylogeny (even though the solution computed may not be the most parsimonious).

Note that a persistent perfect phylogeny is a generalization of a perfect phylogeny, as the latter model represents the case when no character is ever lost. Based on this, another definition of persistent phylogeny is the following.

**Definition 5.6** (persistent phylogeny)**.** A persistent phylogeny $T$ is a *perfect* phylogeny if there are no persistent characters.

A fundamental contribution of [66], building upon [47, 67], is to restate the problem of computing a persistent perfect phylogeny as a graph theory question. In the following, to keep the discussion self-contained we briefly recall two pivotal graphs used in this approach. A graph without edges is called *edgeless*. A connected component is called *nontrivial* if it has more than one vertex.

The first graph, called *conflict graph* and defined in Def. 5.8 is based on the notion of *conflicting characters*.

**Definition 5.7** (conflicting characters)**.** Let $M$ be a binary matrix and let $c_1$, $c_2$ be two characters of $M$. Then the configurations induced by the pair $(c_1, c_2)$ in $M$ is the set of ordered pairs $(M[s, c_1], M[s, c_2])$ over all species $S$. Two characters $c_1$ and $c_2$ of $M$ are *conflicting* if and only if the columns $c_1$ and $c_2$ induce the four configurations $(0, 1)$, $(1, 1)$, $(1, 0)$, $(0, 0)$.

**Definition 5.8** (conflict graph)**.** The *conflict graph* $G_c = (C, E_c \subseteq C \times C)$ of a matrix $M$ has vertices $C$ and edges $E_c$ the pairs $(c_i, c_j)$ of conflicting characters.

See Figure 5.6 for an example of conflict graph.

The second graph, called *red black graph*, provides a graph representation a partial, or full, completion of an extended matrix $M_e$.

**Definition 5.9** (red black graph)**.** Let $M_e$ be an extended matrix associated with the input matrix $M$. The red–black graph $G_{RB}$ for $M_e$ is an edge colored graph $(V, E)$ where $V = C \cup S$, given $S = \{s_1, ..., s_n\}$ and $C = \{c_1, \ldots, c_m\}$ the set of species and characters of $M_e$ (i.e., for each two conjugate characters $c^+$ and $c^-$, only $c$ is a vertex of $G_{RB}$). The set of edges $E$ is defined as follows:

i. a pair $(s, c)$ is a black edge iff the corresponding pair $(c^+, c^-)$ at row $s$ is completed as $M_e[s, c^+] = 1$ and $M_e[s, c^-] = 0$;

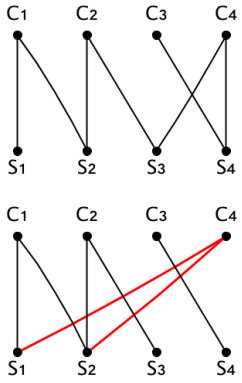
ii. $(s, c)$ is a red edge if the conjugate pair $(c^+, c^-)$ at row $s$ is completed as $M_e[s, c^+] = M_e[s, c^-] = 1$;

iii. $(s, c)$ is not an edge in $G_{RB}$ iff the conjugate pair $(c^+, c^-)$ at row $s$ is completed either $M_e[s, c^+] = M_e[s, c^-] = 0$ or $M_e[s, c^+] = M_e[s, c^-] =$ ?.

Given a red black graph $G_{RB}$ associated to a partial completion of an extended matrix $M_e$, the completion of pairs of conjugate characters in $M_e$ is translated in a graph operation on the edges of the $G_{RB}$, called *realization of a character*, which consists of adding red edges, removing black or red edges. This graph operation over edges of $G_{RB}$ may be iterated till the graph has no edge, i.e., $G_{RB}$ is edgeless.

Let $(c^+, c^-)$ be two conjugate characters of $M_e$. Let $\mathcal{C}(c)$ be the connected component of $G_{RB}$ containing the vertex $c$. A character $c$ in $G_{RB}$ is in one of three possible states:

- *inactive*: $c$ has black incident edges in $G_{RB}$;




| | $c_1^+$ | $c_1^-$ | $c_2^+$ | $c_2^-$ | $c_3^+$ | $c_3^-$ | $c_4^+$ | $c_4^-$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | 1 | 0 | ? | ? | ? | ? | ? | ? |
| $s_2$ | 1 | 0 | 1 | 0 | ? | ? | ? | ? |
| $s_3$ | ? | ? | 1 | 0 | ? | ? | 1 | 0 |
| $s_4$ | ? | ? | ? | ? | 1 | 0 | 1 | 0 |

| | $c_1^+$ | $c_1^-$ | $c_2^+$ | $c_2^-$ | $c_3^+$ | $c_3^-$ | $c_4^+$ | $c_4^-$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | 1 | 0 | ? | ? | ? | ? | 1 | 1 |
| $s_2$ | 1 | 0 | 1 | 0 | ? | ? | 1 | 1 |
| $s_3$ | ? | ? | 1 | 0 | ? | ? | 1 | 0 |
| $s_4$ | ? | ? | ? | ? | 1 | 0 | 1 | 0 |

FIGURE 5.7: The figure illustrates the realization of character $c_4$ in the red-black graph $G_{RB}$ associated with an extended matrix $M_e$. The corresponding completion of the conjugate columns $(c_4^+, c_4^-)$ in $M_e$ is highlighted in red.

- *active*: $c$ has red incident edges in $G_{RB}$;

- *free*: the character $c$ is an isolated node in $G_{RB}$.

Hence, let us present the definition of realization of a character in $G_{RB}$.

**Definition 5.10** (realization of a character)**.** Let $G_{RB}$ be a red-black graph and let $C(c)$ be the set of species that are in the connected component of $G_{RB}$ containing character $c$. The realization of $c$ in $G_{RB}$ corresponds to completing the conjugate pairs $(c^+, c^-)$ in $M_e$ and consists of the following steps:

1. If $c$ is inactive then:

   (a) for each species $s \notin \mathcal{C}(c)$, pose $M_e[s, c^+] = M_e[s, c^-] = 0$;

   (b) for each species $s \in \mathcal{C}(c)$ if there is no black edge $(c, s)$ in $G_{RB}$, add a red edge $(c, s)$ and complete $M_e$ by setting $M_e[s, c^+] = M_e[s, c^-] = 1$;

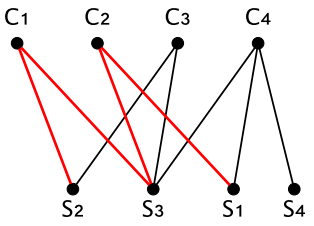
   (c) remove from $G_{RB}$ all black edges $(c, s)$ and label $c$ active.

2. Else if $c$ is active and $c$ is connected by red edges to all species in $\mathcal{C}(c)$, then:

   (a) remove all red edges $(s, c)$ from $G_{RB}$ and label $c$ free;

Fig. 5.7 shows an example of realization of a character in a $G_{RB}$ and the corresponding completion of its conjugate pairs in the associated extended matrix $M_e$.

Observe that when $c$ is free in $G_{RB}$, the corresponding conjugate pairs of columns $(c^+, c^-)$ in $M_e$ have been completed. Note also that given the input matrix $M$ and its original incomplete extended matrix $M_e$, the corresponding $G_{RB}$ has only black edges, where all characters are inactive. Moreover, a character $c$ in $G_{RB}$, that has been realized, is either




(a) red Σ-graph in $G_{RB}$

| $M_e$ | $c_1^+$ | $c_1^-$ | $c_2^+$ | $c_2^-$ | $c_3^+$ | $c_3^-$ | $c_4^+$ | $c_4^-$ |
|---|---|---|---|---|---|---|---|---|
| $s_1$ | 1 | 0 | 1 | 1 | ? | ? | 1 | 0 |
| $s_2$ | 1 | 1 | 1 | 0 | 1 | 0 | ? | ? |
| $s_3$ | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| $s_4$ | 1 | 0 | 1 | 0 | ? | ? | 1 | 0 |

(b) Completion of $M_e$ which contains a forbidden sub-matrix

FIGURE 5.8: Figure 5.8(a) shows a red black graph that has a red Σ-graph induced by two characters $c_1$ and $c_2$ and three species $s_1, s_2, s_3$. Figure 5.8(b) shows the corresponding completion of the extended matrix $M_e$ associated to $G_{RB}$. The forbidden sub matrix $(0,1),(1,0),(1,1)$, highlighted in red, shows that the corresponding completion of the extended matrix $M_e$ does not admit a perfect phylogeny.

free, or is active but there exists a species $s \in \mathcal{C}(c)$ that is not connected to $c$ by a red edge. In the latter case the realization of $c$ in $G_{RB}$ is *impossible* meaning that $c$ cannot become free, till it is eventually connected to all species in $\mathcal{C}(c)$ by red edges.

In the following we recall from [66] an important definition of subgraph of $G_{RB}$ that it is useful to understand if a partial, or full, completion of an extended matrix is solved by a perfect phylogeny.

**Definition 5.11** (red Σ-graph)**.** Given a red black graph $G_{RB}$, a red Σ-graph is a sub-graph of $G_{RB}$ induced by two characters and three species consisting in a path of four red edges.

**Observation 3.** *Note that the red Σ-graph represents the forbidden matrix* $(0,1),(1,0)$ *and* $(1,1)$ *in a completion of the extended matrix* $M_e$*. Hence, whenever the red Σ-graph is present in the red-black graph, the corresponding completion of* $M_e$ *does not admit a directed perfect phylogeny [45]. Consequently, the input matrix* $M$ *for the PPP problem cannot be solved by a persistent perfect phylogeny. In fact, by definition of red-black graph associated with a partial or full completion of* $M_e$*, a red Σ-graph (that includes two characters* $c_1$ *and* $c_2$ *and three species* $s_1, s_2, s_3$*) corresponds to two columns* $c_1^-, c_2^-$ *in the extended matrix such that have the configurations* $(0,1),(1,1)$ *and* $(1,0)$ *for the rows* $s_1, s_2, s_3$ *in* $M_e$*. Note that in* $G_{RB}$ *characters* $c_1$ *and* $c_2$ *cannot become free and their realization is impossible.*

Fig. 5.8 shows an example of a $G_{RB}$ with a red $\Sigma$-graph and the corresponding completion of $M_e$ containing the forbidden sub-matrix $(0,1),(1,0),(1,1)$.

Now we are ready to describe the main ingredients of the approach (presented in [66]) to compute a persistent perfect phylogeny for an input matrix $M$, if such a tree exists.

### 5.4.3 An algorithm to compute a persistent perfect phylogeny

Recall that the aim is computing a completion $M_f$ of the extended matrix $M_e$ that admits a perfect phylogeny obtained by the realization of a special sequence of characters of the red–black graph $G_{RB}$, called *successful reduction* and defined below.

**Definition 5.12** (successful reduction)**.** Given a graph $G_{RB}$ for an extended matrix $M_e$, a successful reduction of $G_{RB}$ is an ordering $r = \langle c_{i1}, \ldots, c_{im} \rangle$ of the set of characters $\{c_1, \ldots, c_m\}$ of the input matrix $M$ such that $G_{RB}$ becomes an edgeless red-black graph by realizing each character in the sequence $r$ and each active character whenever it can be labeled free.

In [66], it has been shown that finding a solution for an instance of the IP-PP problem is equivalent to computing the existence of a successful reduction for the red–black graph. More precisely, let $M_e$ be an instance of the IP-PP problem. In the following, Theorem 5.13 states that if $M_e$ admits a completion $M_f$ that has a perfect phylogeny $T$, then there exists a successful reduction of graph $G_{RB}$. Vice versa, a successful reduction of the red black graph $G_{RB}$ for $M_e$ provides a completion $M_f$ of $M_e$ that admits a perfect phylogeny, thus giving a solution for the IP-PP instance.

**Theorem 5.13.** *Let $M_e$ be an extended matrix associated with a binary matrix $M$. Then $M_e$ admits a perfect phylogeny, if and only if there exists a successful reduction of the graph $G_{RB}$ for $M_e$.*

The algorithm in [66], based on Theorems 5.5 and 5.13, builds a decision tree $\mathcal{T}$ that explores all permutations of the set $C$ of characters of $M_e$ (i.e., for each conjugate pair $(c^+, c^-)$, only the corresponding character $c$ is in $C$) in order to find one that is a successful reduction for $G_{RB}$, if it exists. More precisely, each edge of the decision tree represents a character of $C$ and each path of the tree from the root to a leaf is a distinct permutation of the set $C$. The algorithm works in a branch and bound like manner, in the sense that if a branch of the decision tree ending in node $x$ does not lead to a solution, then the decision tree below $x$ is discarded. More precisely, each branch ending in node $x$ gives a partial permutation $\pi$ that consists of all characters labeling the path from root $r$ of $\mathcal{T}$ to node $x$. A partial completion $M_\pi$ is computed by realizing the

sequence of characters, provided by the partial permutation $\pi$, in the red black graph $G_{RB}$. Whenever $M_\pi$ contains the forbidden matrix, then the branch ending in $x$ does not lead to a solution, and $x$ is labeled as a fail node.

The worst case time of the algorithm is achieved when the whole decision tree $\mathcal{T}$ is explored. Generating all permutations requires $m!$ time. In [66] the authors showed that the time complexity is $O(m! \times n \times m \times \alpha(n^2 \times m))$, improving the complexity of a trivial algorithm that tries all possible completions for the pairs $(?, ?)$, and would require a worst time that is exponential in both the number of species and columns of the input matrix.

In Chapter 6, we consider the computational problem of explaining binary data by the persistent perfect phylogeny model (referred as PPP problem) and investigating its computational complexity. For this purpose we investigate the problem of reconstructing a persistent phylogeny where some constraints are imposed on the paths of the tree.

# Chapter 6

# Explaining evolution via constrained persistent perfect phylogeny

In this chapter we define a natural generalization of the Persistent Perfect Phylogeny (PPP) problem by combining the PPP and the Generalized Cladistic Character Compatibility (GCCC) problems. We introduce the Constrained Persistent Perfect Phylogeny problem (CPPP) which generalizes the PPP problem by adding constraints for some characters in the input matrix [5].

We explore some algorithmic solutions for the CCCP problem. More precisely, based on a graph formulation of the CPPP problem, we are able to provide a polynomial time solution for matrices whose conflict graph has no edges. Using this result, we also develop a parameterized algorithm for solving the CPPP problem where the parameter is the number of characters, partially answering the open problem of determining the computational complexity of the PPP problem.

We conduct a preliminary experimental analysis, both on simulated and real data, showing that our method can manage successfully binary characters data incorporating back mutations.

Finally, we give a characterization of binary matrices that cannot be described by the persistent perfect phylogeny model.

## 6.1 The constrained persistent perfect phylogeny

We now formally define the constrained persistent perfect phylogeny (CPPP) problem. The input of the problem is a binary matrix $M$ and a set $F = \{(c_{i_1}, s_{i_1}), \ldots, (c_{i_l}, s_{i_l})\}$ of constraints, such as $M[s_{i_j}, c_{i_j}] = 0$ for each $j$. The fact that a pair $(c, s)$ (i.e., a character $c$ and a species $s$) is constrained means that $s$ and all of its ancestors do not have the character $c$. In other words, $c$ cannot be persistent on the path from the root of the tree to the leaf node $s$. A solution for this instance is a persistent perfect phylogeny $T$ for $M$ such that, for each constraint $(c_{i_j}, s_{i_j})$, none of the edges from the root of $T$ to the leaf labeled by $s_{i_j}$ is labeled $c^+_{i_j}$. This implies that no edge from the root of $T$ to the leaf labeled by $s_{i_j}$ can be labeled $c^-_{i_j}$.

The idea of the extended matrix $M_e$ applies also to the CPPP problem. In this case, if $M[s, c] = 1$, then $M_e[s, c^+] = 1$ and $M_e[s, c^-] = 0$, if $M[s, c] = 0$ and $(c, s)$ is a constraint, then $M_e[s, c^+] = M_e[s, c^-] = 0$. Finally, if $M[s, c] = 0$ but $(c, s)$ is not a constraint, then $M_e[s, c^+] =?$ and $M_e[s, c^-] =?$. An immediate extension of the result in [66] shows that $M_e$ is solved by a directed perfect phylogeny if and only if $(M, F)$ is solved by a constrained persistent perfect phylogeny.

Just as for the PPP problem, we explore a graph formulation of the CPPP problem based on the equivalence of PPP to the problem of completing an extended matrix $M_e$ associated with a binary matrix $M$. The graph formulation derives again by representing a completion in terms of the red black graph associated with an extended matrice, as described in Section 5.4.2.

Recall that there exists a 1-to-1 correspondence between completing entries of an extended matrix $M_e$ and realizing characters in the corresponding the red black graph $G_{RB}$. When considering the CPPP problem, some entries of a partially completed matrix are constrained meaning that some characters in the associated red black graph cannot be realized. On the contrary, all characters in a red black graph for the PPP problem can be realized. Thus, it is quite easy to show that the main red black graph reduction characterization stated for the PPP problem can be extended to the constrained persistent perfect phylogeny problem, by simply adding the constraint that some characters cannot be realized in the red black graph.

The red black graph reduction turns out to be quite useful to investigate new algorithmic solutions for the PPP problem. In [5] we are able to prove that there exists a class of binary matrices that always admit a solution for the PPP problem. In other words, this kind of instances admits a persistent perfect phylogeny that can be computed in polynomial time. Hence, we provide a polynomial time algorithm for this special case. Based on the latter algorithm we give a fixed-parameter algorithm for the CPPP problem,

where the parameter is the number of characters. This algorithm is based on the search tree technique [68], improving the exponential time algorithm given in [66] and described in Section 5.4.3.

We observe that the CPPP problem is a special case of the General Character Cladistic Compatibility problem (GCCC) [64]. An instance of the GCCC problem is a matrix $M_G$ where rows are species and columns are characters. Each entry of the matrix $M_G$ is a subset of the states that character $c$ may assume in species $s$. Another part of the instance is a specification of all allowed transitions between states in a solution. A feasible solution is a perfect phylogeny where for each species $s$ and for each character $c$, the state is picked from the input set $M_G[s, c]$. Given an instance $(M, F)$ of CPPP, we obtain a matrix $M_G$ as follows. If $M[s, c] = 1$, then $M_G[s, c] = \{1\}$. If $M[s, c] = 0$ and $(c, s) \in F$, then $M_G[s, c] = \{0\}$. Finally, if $M[s, c] = 0$ and $(c, s) \notin F$, then $M_G[s, c] = \{0, 2\}$. The only allowed transitions are from state 0 to 1 and from 1 to 2. This case of GCCC corresponds to case 6 of Table 1 in [64], whose complexity is reported as open. Thus the results we give in [5] also apply to this case.

Recall that a main result of [66] is that finding a solution of PPP is equivalent to finding a successful reduction for a red black graph $G_{RB}$, that is a sequence of characters whose realization in $G_{RB}$ makes the red black graph edgeless (see Section 5.4.3). For the CPPP problem a similar result holds, but we have to adapt the notion of successful reduction, so that there is a third case when the reduction is impossible: when for some species $s$ with $(c, s) \in F$ (i.e., it must be $M_e[s, c^+] = M_e[s, c^-] = 0$), $(c, s)$ is also a red edge of $G_{RB}$.

In the following section we give a different notion of successful reduction, that we call *c-reduction*, corresponding to a sequence of edge labels in a depth first traversal of a persistent perfect phylogeny.

## 6.2 The c-reduction and the persistent perfect phylogeny

In this section we explore the connections between the notion of persistent phylogeny, the red black graph and a successful reduction.

Let $T$ be a persistent perfect phylogeny for a binary matrix $M$. Consider a depth-first traversal of $T$. The sequence of edge labels $\mathcal{C}$ traversed during the depth fist traversal of $T$ is uniquely defined. The converse also holds, that is given a sequence $\mathcal{C}$ of edge labels, we can reconstruct the unique persistent perfect phylogeny $T$ such that $\mathcal{C}$ is the sequence of edge labels traversed during a depth-first traversal of $T$.

The main idea is that we associate a partial phylogeny $P$ to each prefix of $\mathcal{C}$, where each leaf $x$ of $P$ is labeled with the submatrix $M_x$ of $M_e$ such that $M_x$ has exactly the species and the characters that are in the subtree of $T$ rooted at $x$. Recall that each matrix $M_x$ has a graphical representation given by the red black graph. Then determining the next edge label ($c_i^+$ or $c_i^-$) to be added to the prefix of $\mathcal{C}$ corresponds to realizing the corresponding unsigned character $c_i$ (respectively making $c_i$ active or free) in the red black graph associated with $M_x$, as described in Definition 5.10.

Observe that given a persistent phylogeny $T$ for $M$, we can label each node of $T$ with a red black graph whose vertices are the species and the characters of the extended matrix $M_e$ associated with $M$. Note that for each conjugate pair $(c^+, c^-)$ in $M_e$, only the corresponding unsigned character $c$ of $M$ is a node in $G_{RB}$. The red black graph labeling the root of $T$ (also called the red black graph associated to $M$) contains only black edges $(s, c)$ for each species $s$ and character $c$ such that $M[s, c] = 1$. The red black graph $G_{RB}(w)$ labeling a node $w$ of $T$ is defined recursively from the red black graph $G_{RB}(v)$ labeling the parent $v$ of $w$. Given the correspondence between a partial completion of an extended matrix $M_e$ and a red black graph, the red black graph labeling a node $v$ is a complete representation of the sub-matrix of $M_e$ that must be solved by a subtree rooted at $v$. In other words, while constructing a persistent phylogeny $T$, the operation of adding a new edge to $T$ labeled by $c^+$ or $c^-$ corresponds to realizing the corresponding unsigned character $c$ in the red black graph.

In [6] we give a definition of successful reduction that is slightly different from the one in [66] (see Section 5.4.3).

**Definition 6.1** (c-reduction [6])**.** A c-reduction is a sequence $\mathcal{C}$ of characters, such that each character appears once or twice in $\mathcal{C}$. Moreover, if a negative character $c^-$ appears in $R$, then the corresponding positive character $c^+$ precedes $c^-$ in $\mathcal{C}$.

**Observation 4.** *A fundamental observation is that a traversal of a persistent phylogeny $T$ corresponds to a c-reduction, and that a c-reduction corresponds to a traversal of a persistent phylogeny $T$. Moreover, this correspondence exists also between prefixes of a c-reduction and partial persistent phylogenies. We recall that, to obtain an algorithm for PPP, it suffices to have an algorithm that finds the edge label to be added to the prefix of $\mathcal{C}$ computed up to that point. The c-reduction $\mathcal{C}$ obtained by a depth-first traversal of the tree is a sequence of edge labels whose realization results in an edgeless red black graph.*

This observation allows to describe exponential time algorithms that basically consist of enumerating all possible c-reductions, exploiting the fact that if a given partial c-reduction leads to a realization with an edgeless red black graph, then it is impossible to extend

| $M$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ |
|---|---|---|---|---|---|
| $s_1$ | 0 | 0 | 0 | 1 | 0 |
| $s_2$ | 0 | 0 | 1 | 1 | 1 |
| $s_3$ | 0 | 1 | 1 | 0 | 0 |
| $s_4$ | 1 | 1 | 0 | 0 | 0 |
| $s_5$ | 1 | 1 | 1 | 0 | 1 |

FIGURE 6.1: Binary matrix $M$ of size $5 \times 5$

the partial c-reduction into a complete c-reduction (that is a persistent phylogeny solving the original instance).

In the following, we give a definition of conflict graph $G_c$ associated with a generic red black graph $G_{RB}$ rather than with a binary matrix $M$, as described below.

**Definition 6.2** (conflict graph associated with a red black graph)**.** Given an input binary matrix $M$ and a red black graph $G_{RB}$ associated with a partial completion of the extended matrix $M_e$ of $M$, we compute the sub-matrix $M'$ of $M$ by selecting only species and characters that are not isolated nodes in $G_{RB}$ (i.e., they are at least in one connected component of $G_{RB}$). The conflict graph $G_c$ associated with $G_{RB}$ has nodes that are characters in the sub-matrix $M'$ and there is an edge $(c_i, c_j)$ in $G_c$ iff $c_i$ and $c_j$ are conflicting in $M'$(see Definition 5.7).

### An example of correspondence between successful of c-reduction and a persistent perfect phylogeny $T$

Given the binary matrix $M$ in Fig. 6.1, Fig. 6.2 illustrates a persistent perfect phylogeny $T$ that explains $M$.

Observe that the sequence of edge labels $\mathcal{C} = \langle c_4^+, c_3^+, c_5^+, c_2^+, c_4^-, c_1^+, c_5^-, c_3^-, c_1^- \rangle$, traversed by a the depth-first traversal of $T$, corresponds to a successful c-reduction of the red black graph $G_{RB}$. More precisely, the root $r$ of $T$ is labeled with the initial red black graph $G_{RB}(r)$ having only black edges and no active character. Given the edge $(r, v)$ labeled $c_4^+$ in $T$, $v$ is labeled with the red black graph $G_{RB}(v)$ obtained after the realization of the inactive character $c_4$ in $G_{RB}(r)$ as we can see in Fig. 6.2. Similarly, given the edge $(w, z)$ with label $c_4^-$, $w$ is labeled with the red black graph $G_{RB}(w)$ where $c_4^-$ is active and not free, while $z$ is labeled with the red black graph $G_{RB}(z)$ obtained from $G_{RB}(w)$ by making $c_4$ free ($c_4$ becomes an isolated node in $G_{RB}(z)$). This holds for all the edges in $T$ while traversing $T$ by means of a depth-first traversal. Note that the last edge that is traversed, say $(y, l)$, is the one labeled by $c_1^-$, the character that appears as last in $\mathcal{C}$. Observe that the node $l$ is labeled by the edgeless red black graph $G_{RB}(l)$. This shows an example of the fact that a traversal of a persistent phylogeny $T$ corresponds to a

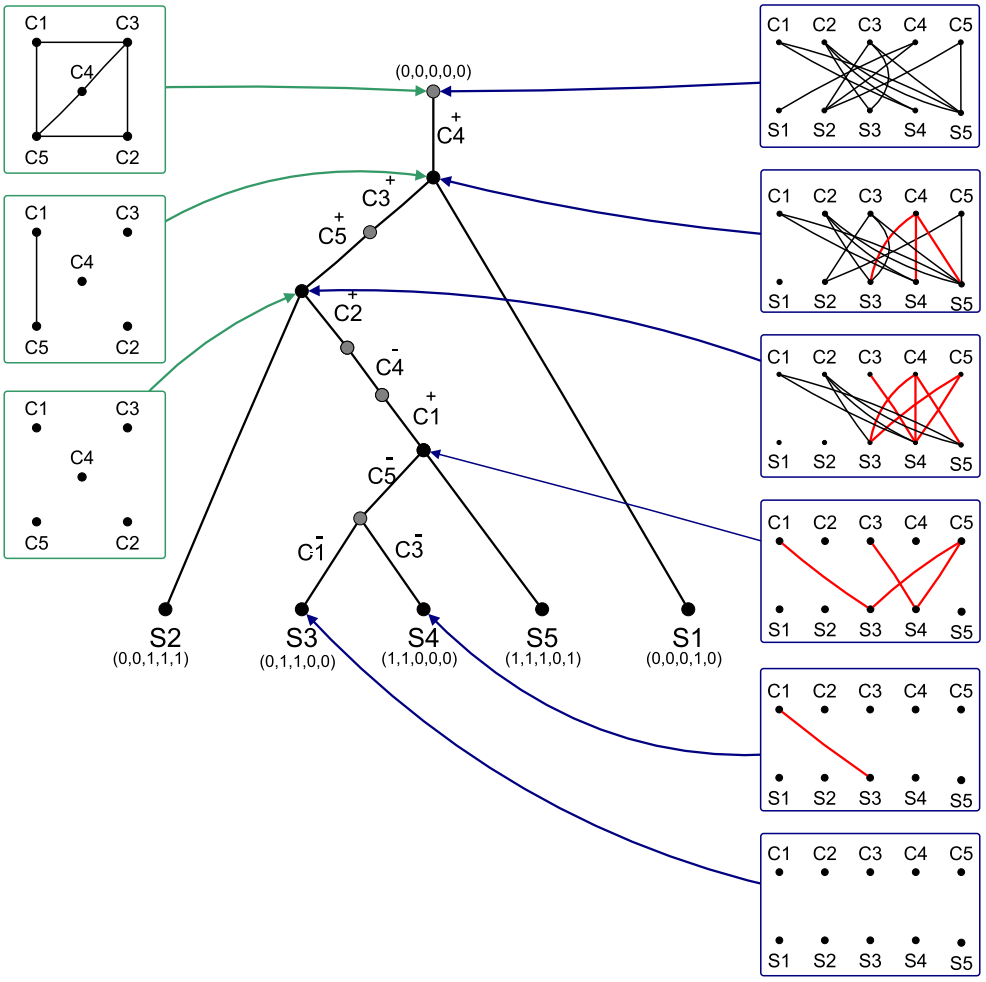



FIGURE 6.2: The figure illustrates that the c-reduction $\mathcal{C} = \langle c_4^+, c_3^+, c_5^+, c_2^+, c_4^-, c_1^+, c_5^-, c_1^-, c_1^- \rangle$ corresponds to a depth-first traversal of a persistent perfect phylogeny for the binary matrix $M$ if Fig. 6.1. Each node of the three has its associated $G_{RB}$. In the figure only some red black graphs are shown. Finally, some nodes of $T$ are labeled with conflict graphs associated with red black graphs labelling the corresponding nodes.

c-reduction, and that a c-reduction corresponds to a traversal of a persistent phylogeny $T$.

Observe that each red black graph labeling a node in $T$ (see Fig. 6.2) has its associated conflict graph. For example the root $r$ of $T$ is labeled with $G_{RB}(r)$ that is the red black graph, where all characters are inactive, associated to the whole binary matrix $M$ in Fig. 6.1. Given the edge $(r, v)$ labeled by $c_4^+$, the node $v$ in $T$ is labeled with the red black graph $G_{RB}(v)$. The latter has associated a conflict graph with only one edge $(c_1, c_2)$. Indeed, by selecting from $M$ only the characters and the species that are non-isolated nodes in $G_{RB}(v)$, we obtain a sub-matrix $M'$ of $M$ that does not have the species $s_1$. Hence, the conflict graph of $M'$ has, hence, only one pair of conflicting characters $(c_1, c_2)$. Similarly, given the edge $(e, d)$ of $T$ labeled by $c_5^+$, the node $d$ is labeled with the red

black graph $G_{RB}(d)$ obtained from $G_{RB}(e)$ after the realization of $c_5^+$ in $G_{RB}(e)$. The conflict graph associated with $G_{RB}(d)$ is edgeless.

## 6.3 Solving CPPP on matrices with edgeless conflict graphs

In the following, we will exploit some properties of the red black graph to demonstrate that the PPP problem for matrices that have an edgeless conflict graph has always a solution. More precisely, we give a polynomial algorithm for solving the PPP problem for such matrices that consists in finding a successful reduction for the associated red black graphs.

First, we need to give other basic definitions that are fundamental for the algorithm we develop: neighborhood of character in $G_{RB}$; maximal characters in a red black graph $G_{RB}$; adjacency graph.

**Definition 6.3** (neighborhood of a character)**.** Given a red black graph $G_{RB}$ and given a character $c$ in $G_{RB}$ the neighborhood of $c$ is the set of species $N(c)$ that are directly connected to $c$ either by black edges or by red edges (dependently from the fact that $c$ is inactive or active in $G_{RB}$).

**Definition 6.4** (maximal character)**.** Given a red black graph $G_{RB}$, a character $c$ is *maximal* in $G_{RB}$ if there is no character $c'$, such that $c \neq c'$ and $N(c) \subseteq N(c')$.

We build a graph $G = (V; E)$, called *adjacency graph* for a red black graph $G_{RB}$, defined as follows.

**Definition 6.5.** (adjacency graph) The adjacency graph $G$ for a red black graph $G_{RB}$, is a pair of $G = (V, E)$, where $V$ is the set of characters of $G_{RB}$ and $(u, v)$ is an edge of $G$ if and only if $u, v$ are *adjacent* in $G_{RB}$, i.e., there is at least one species $s$ such that $s$ is adjacent to both $u$ and $v$ in the red black graph $G_{RB}$. In other words, $u$ and $v$ share at least one species.

Given an instance $(M, F)$ for the CPPP problem, we say that one character $c_k$ of $M$ *can be realized* in $G_{RB}$ if $c_k$ does not belong to any pair $(c_{i_j}, s_{i_j}) \in F$. In other words, $c_k$ can be realized if it is not involved in any constraint.

Our algorithm for solving the PPP and CPPP problems, for instances described by edgeless conflict graphs, finds a successful reduction by simply at each step computing the maximal inactive characters that can be realized in the red black graph following the set of constraints.

We first state some Lemmas that are used to show that maximal characters in $G_{RB}$ can be realized without inducing any red $\Sigma$-graph in the red black graph (see Definition of red $\Sigma$-graph 5.11).

The following property is easily proved by induction on the length of a path in the red black graph connecting two maximal characters.

**Lemma 6.6.** *Let $G_{RB}$ be a connected red black graph such that $G_{RB}$ has no red $\Sigma$-graph and the associated conflict graph $G_c$ is edgeless. Let $C_M$ be the set of maximal inactive characters in $G_{RB}$. Then $C_M$ consists of elements that are pairwise adjacent in the adjacency graph for $G_{RB}$.*

*Proof.* Assume that $M$ is the matrix for the red black graph $G_{RB}$. Let $a, b$ be two arbitrary characters in $C_M$.

Since the red black is connected, there exists a path $\pi$ connecting the two vertices $a$ and $b$. Then by induction on the number $k$ of internal characters of the smallest path in $G_{RB}$ that connects $a$ to $b$ we show that $(a, b)$ is an edge in the adjacency graph $G$. Assume first that $k = 1$, i.e. $c$ is the only internal character of the shortest path $\pi$. It follows that $(a, c)$, $(c, b)$ are edges of the adjacency graph $G$.

Assume on the contrary that there is no edge $(a, b)$ in the adjacency graph. First assume that $c$ is comparable with $b$ or $a$ in $G_{RB}$. Since $a, b \in C_M$, it holds that $c < b$ or $c < a$. But this fact would imply that $a, b$ are adjacent in graph $G$ as they must a common species with $c$, being $G_{RB}$ connected and $c$ adjacent to $a$ or $b$ in the adjacency graph $G$.

Hence, assume now that $c$ is not comparable with both $a$ and $b$. First let us consider the case that $c$ is non active in $G_{RB}$. The following cases must be considered. First, since $a$ is adjacent to $c$, there exists $s_1$ such that $a$ and $c$ share $s_1$, i.e. $s_1 \in N(a) \cap N(c)$. Hence we have the configuration $(1, 1)$ for $(a, c)$ in $M$. Moreover, as $c$ is not comparable with $a$, there exists as species $s_2$, s.t. $s_2 \in N(a) \setminus N(c)$ and vice versa there exists a species $s_3$, s.t. $s_3 \in N(c) \setminus N(a)$. Thus we have the configurations $(1, 0)$ and $(0, 1)$ for the pair of characters $(a, c)$. Now, being $b$ maximal and not comparable with $c$, there must be a species $s_4$ s.t. $s_4 \in N(b) \setminus N(c)$. Being $N(b)$ and $N(a)$ disjoint sets (as we assumed $a$ and $b$ do not share any species), it follows that $s_4 \notin N(a) \cup N(c)$. Hence we have also the fourth configuration $(0, 0)$ for the pair $(a, c)$. But this fact would imply that $(a, c)$ is an edge of the conflict graph $G_c$ which is a contradiction with the hypothesis.

Let us now consider the case that $c$ is an active character connecting $a$ and $b$ in $G_{RB}$ by red edges. Recall that by assumption $c$ is not comparable with both $a$ and $b$.

Let $N(c)^-$ denote the species that are not in $N(c)$. Note that $N(c)^-$ is the set of species connected to $c$ by black edges before its realization in $G_{RB}$.

Now, as $c$ is connected by a red edge to a species $s_1$ possessed by $a$ then $s_1 \in N(a) \setminus N(c)^-$. Hence the pair $(a, c)$ has the configuration $(1, 0)$ for the species $s_1$ in $M$. Similarly, there is a species $s_2 \in N(b) \setminus N(c)^-$, as $s_2$ is possessed by $b$ but there is a red edge connecting $c$ and $s_2$. Note that since $N(a)$ and $N(b)$ are disjoint, $s_2 \notin N(a)$. This means that the pair $(a, c)$ has the configuration $(0, 0)$ for the species $s_2$ in $M$.

Since $c$ and $b$ are not comparable there must be a species $s_3$ such that $s_3 \in N(b)$ and $s_3 \notin N(c)$, then $s_3$ must be in $N(c)^-$. Recall that $N(a)$ and $N(b)$ are disjoint by assumption, meaning that $s_3 \notin N(a)$. Thus the pair $(a, c)$ has the configuration $(0, 1)$ for the species $s_3$ in $M$.

For the same reason, since $a$ and $c$ are not comparable in $G_{RB}$, there must be a species $s_4$ such that $s_4 \in N(a)$ and $s_4 \in N(c)^-$. Hence the pair $(a, c)$ have the configuration $(1, 1)$ for the species $s_4$ in $M$. Consequently, $(a, c)$ are in conflict that is a contradiction with the initial assumption on the conflict graph.

Then assume that $k = n > 1$. Clearly, $G$ must have edge $(a, c)$ where $c$ is the vertex adjacent to $a$ on the path having $k$ internal vertices. Since the path connecting $c$ and $b$ has $k - 1$ internal vertices, by induction it holds that also edge $(c, b)$ is in graph $G$. Consequently, there exists a path with 1 vertex in the red black graph connecting $a$ and $b$. By induction, we show that $(a, b)$ must be an edge of the graph $G$, as required. □

The following properties can be proved as consequence of the definition of realization of characters in the red black graph and assuming that the input matrix is associated with has an edgeless conflict graph.

**Lemma 6.7.** *Let $G_{RB}$ be a connected red black graph has no red $\Sigma$-graph and the associated conflict graph is edgeless. Then the realization of two maximal non active characters $a$ and $b$ in $G_{RB}$ does not produce any red $\Sigma$-graph.*

*Proof.* First note that by Lemma 6.6 $a$ and $b$ are adjacent in $G_{RB}$, thus $(a, b)$ is an edge in the adjacency graph. In other words, there is a species $s_1$, s.t. $s_1 \in N(a) \cap N(b)$. Also, since $a$ and $b$ are maximal characters there must be two species $s_2, s_3$, s.t. $s_2 \in N(a) \setminus N(b)$ and $s_3 \in N(b) \setminus N(a)$. Hence, there does not exist a species $s_4$, such that $s_4 \notin N(a)$ and $s_4 \notin N(b)$, otherwise $(a, b)$ would be an edge in the conflict graph $G_c$ that is a contradiction with the hypothesis. It follows that after the realization of $a$ and $b$ in $G_{RB}$ the red edges adjacent to $a$ are disjoint from the red edges adjacent to $b$. Moreover, the red-edges adjacent to $a$ include the red-edges of active characters contained in $b$ and the same holds for the red-edges adjacent to $b$.

Now, assume that there is a character $c$ active in $G_{RB}$ that is adjacent to $b$. Either $c$ is free after the realization of $b$, otherwise the realization of $b$ adds red-edges that are in

**Algorithm 1:** Procedure Solve-CPPP-edgeless-conflict

**Input** : A red black graph $G_{RB}$ s.t. its associated conflict graph $G_c$ is edgeless and a set $F$ of constraints.

**Output** : A realization $S_c$ of the characters of $G_{RB}$ resulting in a constrained persistent perfect phylogeny taking in account $F$, if such a phylogeny exists.

```
1  S_c ← empty sequence;
2  while G_RB is not edgeless do
3  |   foreach active character c in G_RB that is free do
4  |   |   realize c in G_RB;
5  |   C_M ← maximal inactive characters that are in the same connected component of
   |   G_RB;
6  |   D ← the subset of C_M consisting of the characters that can be realized, i.e., the set
   |   of constrained F is satisfied;
7  |   if D = ∅ then
8  |   |   return no solution
9  |   else
10 |   |   Add to S_c all characters in D;
11 |   |   Realize the characters of D in any order, updating G_RB;
12 |   |   add to D the free characters in the graph G_RB;
13 return S_c;
```

the same red connected component of $c$. The same fact holds after the realization of a character $a$ that is adjacent to $b$ in the adjacency graph. This is a consequence of the fact that as stated above there does not exists species $s, s', s''$ such that $s \notin N(a) \cup N(b)$, $s' \notin N(a) \cup N(c)$ and $s'' \notin N(b) \cup N(c)$. Thus the Lemma follows. □

*Remark* 6.8. The absence of conflicts does not guarantee that a solution for the CPPP problem actually exists, while a solution always exists for the unconstrained PPP problem. However, we are able to provide an efficient algorithm (Algorithm 1) for the CPPP problem in absence of conflicting characters, which is a cornerstone for our algorithm for the general case. Observe that Algorithm 1 can be also applied to find a solution for the PPP problem simply setting $F = \emptyset$.

Algorithm 1 builds a successful reduction $S_c$ by iteratively adding to $S_c$ the maximal inactive characters of the red black graph $G_{RB}$. Recall that the successful reduction in $G_{RB}$ provides a completion of the extended matrix that admits a perfect phylogeny. The latter can be built using the classical linear time algorithm for reconstructing a perfect phylogeny [45].

**Theorem 6.9.** *Let $G_{RB}$ be a connected red black graph such that $G_{RB}$ has no red $\Sigma$-graph and the associated conflict graph is edgeless. Then Algorithm 1 computes a successful reduction of $G_{RB}$.*

*Proof.* We show that at each iteration of Algorithm 1 the following invariant property holds: the connected components of $G_{RB}$ have no red $\Sigma$-graph. Consequently, at the end of the iterative step, when no inactive character is left, it holds that all active characters are free and the graph $G_{RB}$ is edgeless, as required, showing that $S_c$ is a successful reduction.

Clearly, the invariant property holds initially for $G_{RB}$. Moreover, at each iteration the invariant property holds as a consequence of Lemma 6.6 and Lemma 6.7. In fact, only maximal inactive characters are realized, which by Lemma 6.6 are pairwise adjacent. By Lemma 6.7, their realization does not produce any red $\Sigma$-graph. □

## 6.4 An algorithm for CPPP problem

In this section we propose an algorithm for the CPPP problem that extends the algorithm in [66] (discussed in Section 5.4.3), but uses the procedure **Solve-CPPP-edgeless-conflict**$(G_{RB}, F)$ to improve the efficiency. Our algorithm uses implicitly a search tree technique [68] to explore the tree of all possible permutations of characters to be realized in $G_{RB}$. The search tree has at most $m!$ leaves, where $m$ is the number of columns of the input matrix $M$. Therefore we only need to describe a polynomial-time algorithm to compute an edge of the search tree (which mainly consists of realizing a character $c$ of $M$ in $G_{RB}$). Just as the algorithm in [66], we associate to the matrix $M$, of the instance $(M, F)$, an extended matrix $M_e$ which is then analyzed to find a solution. In fact, $(M, F)$ has a solution if and only if there exists a successful reduction for $G_{RB}$ which respects $F$ and corresponds to a completion $M_f$ of $M_e$ explained by a perfect phylogeny. Indeed, we require also that all constraints in $F$ are satisfied for the CPPP problem.

The algorithm in [66] explores all feasible permutations of the set of characters to be realized in $G_{RB}$ in order to find one that is a successful reduction, if such a sequence exists. Clearly computing all permutations is not efficient, therefore we implicitly build a decision tree $\mathcal{T}$, where at each step we fix a character in a given position of the permutation.

More precisely, for each node $x$ of the decision tree, we compute $M_e(x)$, obtained from $M_e$ by realizing the sequence of characters $\pi_x$ labelling the edges on the path from the root of $\mathcal{T}$ to $x$. We label $x$ with the corresponding red black graph $G_{RB}(x)$ obtained realizing $\pi_x$. Then, we compute the conflict graph $G_c(x)$ for $x$ associated with $G_{RB}(x)$ (See Definition 6.2). When $G_c(x)$ is edgeless, instead of further exploring the decision tree, we apply Algorithm 1, hence we cut the decision tree. On the other hand, if $G_{RB}(x)$ contains a red $\Sigma$-graph, then $M_e(x)$ does not admit a perfect phylogeny. In this case $M$

cannot admit a persistent perfect phylogeny, hence we can stop exploring that portion of the decision tree. Moreover, we can stop the search as soon as we find a solution, since we have no optimization criterion to discriminate among feasible solutions. In practice, all those criteria allow to avoid exploring a large part of the decision tree, improving the efficiency of the algorithm in [66] as shown in our experimental analysis.

## 6.5 Experimental analysis

We implemented our algorithm as a C++ program and we tested it over simulated data produced by *ms* [26]. Moreover, we tested our program on real data coming from the International HapMap project [69]. All tests have been performed on a standard workstation.

The two different kinds of data correspond to two separate goals. The analysis on simulated data is aimed at studying the scalability of our approach for increasing numbers of species and characters. More precisely we run our program for $n = 10, 20, 40, 60$ (recall that $n$ is the number of species) and for values of $m$ (the number of characters) ranging from $n/2$ to $\frac{3}{2}n$.

Moreover, *ms* produces matrices that have a perfect phylogeny, but can have duplicated rows and columns. To introduce back mutations, we randomly modified at most one state of each duplicated row. For each choice of the parameters $n$ and $m$ we produced 100 random instances, on which we have run our program with a 15-minute timeout, without imposing any constraint. The results are represented in Table 6.1.

For the first 10 out of the 100 instances of each parameter choice, we modified the input matrices, by introducing some random constraints, in order to settle if constraining the set of feasible solutions can help in finding a persistent phylogeny. For each instance of the first phase, we produced 10 instances with 1 or 16 random constraints. For both cases we determined when at least one of the 10 constrained instances is solved more quickly than the unconstrained instance. The goal is to settle when there is a sizable (in our case 10%) probability that introducing some random constraints can help in computing a persistent phylogeny. Moreover, we determine when the median of the 10 constrained instances is solved more quickly than the unconstrained instance. In this case the goal is to determine when there is a 50% probability that some random constraints can help in computing a persistent phylogeny.

The most important result of this experiment is that for instances where our implementation requires at least a second (on average), the idea of introducing random constraints

| Species | Characters | Instances completed within 15 min. | Min time (sec) | Max time (sec) | Avg. time (sec) | Std. dev. |
|---|---|---|---|---|---|---|
| 10 | 5 | 100/100 | 0 | 0.01 | 0.00 | 0.00 |
| 10 | 7 | 100/100 | 0 | 0.25 | 0.01 | 0.03 |
| 10 | 10 | 100/100 | 0 | 1.93 | 0.11 | 0.30 |
| 10 | 12 | 94/100 | 0 | 12.95 | 0.84 | 1.93 |
| 10 | 15 | 84/100 | 0 | 43.89 | 5.71 | 9.80 |
| 20 | 10 | 100/100 | 0 | 4.72 | 0.08 | 0.47 |
| 20 | 15 | 97/100 | 0.02 | 18.12 | 1.15 | 2.53 |
| 20 | 20 | 93/100 | 0.13 | 95.03 | 10.44 | 19.14 |
| 20 | 25 | 79/100 | 1.09 | 253.68 | 41.98 | 60.35 |
| 20 | 30 | 63/100 | 3.84 | 247.03 | 59.06 | 63.81 |
| 40 | 20 | 100/100 | 0.06 | 89.02 | 2.04 | 8.93 |
| 40 | 30 | 98/100 | 0.99 | 156.16 | 22.03 | 33.17 |
| 40 | 40 | 80/100 | 7.23 | 598.32 | 128.47 | 154.92 |
| 40 | 50 | 45/100 | 19.14 | 585.42 | 198.81 | 146.39 |
| 40 | 60 | 19/100 | 50.26 | 577.1 | 319.25 | 183.10 |
| 60 | 30 | 99/100 | 0.64 | 222.79 | 14.36 | 33.21 |
| 60 | 45 | 90/100 | 8.76 | 590.03 | 123.05 | 148.48 |
| 60 | 60 | 51/100 | 37.63 | 593.06 | 252.34 | 168.92 |

TABLE 6.1: Running times on unconstrained simulated instances. All times are in seconds.

is often beneficial. This fact suggests a direction for further improvements, that is incorporating into our program some deterministic constraints, based on a cursory analysis of conflict graphs and red black graphs. Actually, how we manage an edgeless conflict graph is as an example of this idea. Table 6.2 summarizes the experiment on constrained simulated instances.

Finally, the algorithm has been tested on real data coming from the International HapMap project [69]. The data are classified by type of population. In our case, we used data from the set ASW (African ancestry in Southwest USA). Each individual is described by the two haplotypes (in our application the two haplotypes correspond to two different species, i.e. two different rows of the matrix). This experiment investigates the usefulness of the constrained persistent model to manage haplotypes data that cannot be explained by the perfect phylogeny model. In fact, none of those instances admits a perfect phylogeny, but our model and implementation are able to find a reasonable interpretation for the data. The data set consists of binary matrices of dimensions $10 \times 10$, $26 \times 15$, $26 \times 25$, and $26 \times 30$. For each group we considered 10 matrices. In all cases the matrices do not admit perfect phylogeny, and the number of conflicts changes from a minimum of 4 to a maximum of 138. Increasing the size of the matrix, and therefore the number of conflicts, the percentage of matrices that admit persistent perfect phylogeny decreases. More in detail, 80% of the tested matrices of size $10 \times 10$ admits solution, only 20% of the tested matrices of size $26 \times 15$ admits solution, and none of the sets $26 \times 25$, and

$26 \times 30$ admits solution. The results show that haplotype data may be related by the persistent phylogeny in case they cannot be explained by the perfect model. It would be interesting to investigate the biological soundness of the persistent perfect phylogeny in this context.

## 6.6 A characterization of matrices without a persistent phylogeny

In this section we investigate some properties of matrices that do not admit a persistent phylogeny. In the case of the perfect phylogeny, there exists a well known characterization of matrices that do not have a solution as those matrices that contain a small fixed forbidden sub-matrix.

We will show that, differently from the case of the perfect phylogeny, in the persistent case there exists an infinite family of minimal forbidden sub-matrices. More precisely, we

| | | Number of added constraints | | | |
|---|---|---|---|---|---|
| | | 1 | | 16 | |
| Species | Characters | Fastest | Median | Fastest | Median |
| 10 | 5 | 0 | 0 | 0 | 0 |
| 10 | 7 | 1 | 0 | 1 | 1 |
| 10 | 10 | 7 | 5 | 7 | 7 |
| 10 | 12 | 7 | 5 | 7 | 6 |
| 10 | 15 | 8 | 3 | 9 | 8 |
| 20 | 10 | 9 | 4 | 10 | 10 |
| 20 | 15 | 10 | 9 | 10 | 10 |
| 20 | 20 | 9 | 1 | 10 | 10 |
| 20 | 25 | 9 | 7 | 9 | 9 |
| 20 | 30 | 7 | 2 | 10 | 9 |
| 40 | 20 | 9 | 7 | 10 | 10 |
| 40 | 30 | 10 | 7 | 10 | 10 |
| 40 | 40 | 8 | 1 | 10 | 9 |
| 40 | 50 | 10 | 0 | 10 | 10 |
| 40 | 60 | 1 | 0 | 9 | 6 |
| 60 | 30 | 8 | 7 | 10 | 10 |
| 60 | 45 | 10 | 8 | 10 | 10 |
| 60 | 60 | 7 | 6 | 8 | 7 |

TABLE 6.2: Improvements of constrained simulated instances over unconstrained instances. For each choice of the number of species and of characters, we state the number of instances where at least one of the 10 random constrained instances is solved more quickly than the unconstrained instance (columns labeled Fastest). Moreover we state the number of instances where the median of the 10 random constrained instances is solved more quickly than the unconstrained instance (columns labeled Median).

will provide a set of $n \times n$ matrices that cannot be explained by a persistent phylogeny, but whose sub-matrices all have a persistent phylogeny. The proof of these result is based on the red black graph representation of those instances.

**Theorem 6.10.** *Let $M$ be the binary matrix with $n$ rows and $n$ columns (with $n > 4$) such that such that its associated red black graph consists of a simple cycle. Then $M$ does not admit a persistent perfect phylogeny, but any sub-matrix of $M$ admits a persistent phylogeny.*

*Proof.* Assume that $s_1, c_1, s_2, c_2, s_3, c_3 \ldots, s_n, c_n, s_1$ is the simple cycle of the red black graph, where $n > 4$. Then we can show that any permutation of characters the cannot be a successful reduction of the red black graph since the consecutive realization of any pair of characters induces a red $\Sigma$-graph in the $G_{RB}$.

Let $c_i$ and $c_j$ be two characters of the simple cycle, where $1 \leqslant i < j \leqslant n$. Now, each character $c_i$ and $c_j$ is adjacent to two species. But since there are at least 5 species in the simple cycle, it follows that there is a species $s^{'}$ that is not adjacent to both $c_i$ and $c_j$. First notice that $c_i$ is adjacent to species $s_i$ and $s_{i+1}$, $c_j$ for $j \neq n$ is adjacent to $s_j$ and $s_{j+1}$ while, when $j = n$, $c_j$ is adjacent to $s_j$ and $s_1$ closing the cycle. Thus, the consecutive realization of $c_i$ and $c_j$ in $G_{RB}$ produces a red $\Sigma$-graph induced by the pair of characters $(c_i, c_j)$ and the species: (a) $s_i, s^{'}, c_{j+1}$ when $i \geqslant 1$ and $j < n$, or (b) species $s^{'}, s_i, s_1$ when $i > 1$ and $j = n$, or (c) species $s^{'}, s_{i+1}, s_n$ when $i = 1$ and $j = n$.

Recall that the presence of a red $\Sigma$-graph in $G_{RB}$ corresponds to a completion of the extended matrix $M_e$ that does not admit a perfect phylogeny. Finally, since the realization of any pair of characters in the red black graph induces a red $\Sigma$-graph no solution exists for the instance $M$ of the PPP problem.

Now we will prove that any sub-matrix of $M$ has a persistent phylogeny. By construction of $M$, it suffices to prove the cases when the last column or the last row of $M$ is removed. Let $M_r$ be the sub-matrix of $M$ obtained by removing the n-th row. The phylogeny consisting of one single path whose edges are labeled by the sequence of characters (c-reduction) $\langle c_n^+, c_1+, c_n^-, c_2^+, c_1^-, c_3^+, c_2^-, \ldots, c_{n-1}^+ \rangle$ solves the matrix $M_r$. Let $M_c$ be the sub-matrix of $M$ obtained by removing the n-th column. In this case, the phylogeny consisting of one single path whose edges are labeled by the sequences of characters (c-reduction) $\langle c_1^+, c_2+, c_1^-, c_3^+, c_2^-, \ldots, c_{n-2}^-, c_{n-1}^- \rangle$ is a solution for the matrix $M_c$, completing the proof. □

An example of $G_{RB}$ consisting of a simple cycle and associated to an input binary matrix $M$ of size $n \times n$ is in Fig. 6.4.

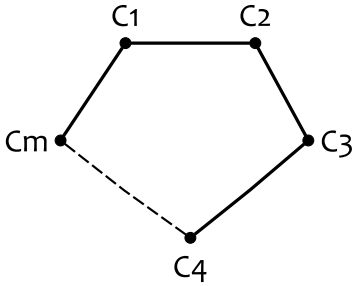



FIGURE 6.3: The figure illustrates the conflict graph $G_c$ associated with the binary matrix $M$ of size $n \times n$ of Lemma 6.10.

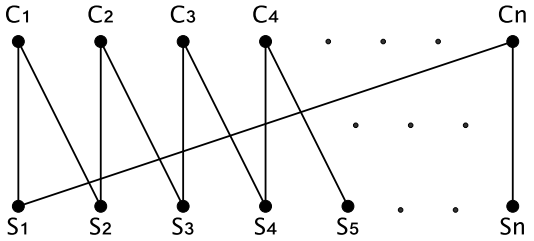



FIGURE 6.4: The figure illustrates the red black graph (a simple cycle) associated with the binary matrix $M$ of size $n \times n$ of Lemma 6.10.

| $M$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ |
|---|---|---|---|---|---|
| $s_1$ | 0 | 0 | 0 | 1 | 1 |
| $s_2$ | 0 | 0 | 1 | 0 | 0 |
| $s_3$ | 0 | 1 | 1 | 1 | 0 |
| $s_4$ | 1 | 0 | 1 | 1 | 1 |
| $s_5$ | 1 | 1 | 0 | 0 | 1 |

FIGURE 6.5: Binary matrix $M$ of size $5 \times 5$ that does not admit a persistent perfect phylogeny

Observe that the set of matrices described in Theorem 6.10 have associated a chordal conflict graph (an example in Fig. 6.3).

Moreover, notice that the forbidden condition stated in Lemma 6.10 is not sufficent to identify matrices that do not have a solution. In fact the matrix $M$ of size $5 \times 5$ illustrate in Fig. 6.5 is another example of matrix that does not have a solution but whose red black graph is not a simple cycle.

# Chapter 7

# Filling incomplete genomic sequences

In this chapter we consider two combinatorial problems that aim to reconstruct complete genomes by inserting a collection of missing genes in incomplete genomic sequences. The next-generation sequencing technologies provide a huge amount of data, that can be used, for instance, for reconstruction studies. However, these technologies are often not able to reconstruct complete genomes and they provide only an incomplete information.

The first problem we consider, called *One-sided scaffold filling*, given an incomplete genome $B$ and a complete genome $A$, asks for the insertion of missing genes into an incomplete genome $B$ with the goal of maximizing the common adjacencies between genomes $B'$ (resulting from the insertion of missing genes in $B$) and $A$. The second problem, called *Two-sided scaffold filling*, given two incomplete genomes $A$, $B$, asks for the insertion of missing genes into both genomes so that the resulting genomes $A'$ and $B'$ have the same multi-set of genes and the number of common adjacencies between $A'$ and $B'$ is maximized.

Both problems were proved to be NP-hard, while their parameterized complexity, when the parameter is the number of common adjacencies of the resulting genomes, was left as an open problem. We settle this open problem by presenting fixed-parameter algorithms for *One-sided scaffold filling* and *Two-sided scaffold filling* problems.

## 7.1 Background and motivations

Comparative genomics is a widely investigated field of bioinformatics in which the genomic features of different organisms are compared in order to identify biological differences and

similarities. The genomic features include DNA sequences, genes, regulatory sequences and other genomic structural landmarks [70]. The ultimate goal of the approaches in this field is to understand genome functions, relationships between organisms and their evolutionary history. In this context, several interesting combinatorial problems have been introduced and studied by the computer science community (see for example [71]).

The introduction of Next Generation Sequencing (NGS) technologies lead to a huge increase on the amount of DNA/RNA and protein sequences available for genomic and trascriptomic analyses [72]. NGS technologies produce millions of short DNA/RNA fragments, called reads, that are joined together to reconstruct longer sequences. While NGS technologies generate such a huge amount of data, the cost of obtaining a complete genome is still high, in particular if compared to the cost of sequencing. Due to this fact, often released genomes are unfinished and incomplete [72]. When used in genomic analyses, incomplete draft genomes (also called *scaffolds*) may introduce errors. Hence, a relevant problem for genome comparison is the filling of scaffolds with missing genes, by mean of combinatorial algorithms, in order to reconstruct complete genomes that share a high level of similarities with a known reference genome.

A combinatorial problem that has been introduced recently is the *One-sided scaffold filling problem* [73]. Such problem consists of filling a scaffold $B$ so that the resulting complete genome $B'$ minimizes the Double-Cut and Join (DCJ) distance [74] with respect to the reference genome $A$. Given two genomes, the DCJ distance is the minimum number of allowed rearrangement operations that transform one genome into the other. The authors presented a polynomial-time algorithm for the problem when the input genomes do not contain duplicated genes.

Later in [75], the scaffold filling problem has been investigated considering both the DCJ distance and the *breakpoint distance*. Given two related sequences $A$ and $B$, two consecutive elements $a_i$ and $a_{i+1}$ in $A$ form an *adjacency* if they are also consecutive in $B$ independently from the order (i.e., as $a_i a_{i+1}$ or $a_{i+1} a_i$), otherwise they form a *breakpoint*. Therefore, the breakpoint distance between $A$ and $B$ is defined as the number of breakpoints in $A$, which is equal to that of $B$. In [75] Jiang et al. introduced a new related variant of the combinatorial problem, called *Two-sided scaffold filling problem*, where both genomes are incomplete. The authors in [75] show that when the input genomes do not contain gene repetitions the problem is polynomially solvable under both the DCJ distance and the breakpoint distance. However, when genomes contain duplicated genes, the scenario changes. Indeed, the authors show that the One-sided problem is NP-complete even under the breakpoint distance.

In [8] we consider a different similarity measure to compare genomes, namely *the maximum number of common adjacencies* between two genomes. This measure has been introduced

for the One-sided/Two-sided scaffold filling problems in [76]. The two problems were both proved to be NP-hard under this similarity measure [77]. The same paper has investigated the approximation complexity of the two problems, showing a 2-approximation algorithm for the Two-sided scaffold filling problem, and a $\frac{4}{3}$-approximation algorithm for the One-sided scaffold filling problem. This latter result has been recently improved in [78], where an approximation algorithm of factor $\frac{5}{4}$ for the One-sided scaffold filling problem has been presented. An approximation algorithm for a related variant of the problem has been given in [79].

In [8], we focus on the parameterized complexity of One-sided scaffold filling and Two-sided scaffold filling. Parameterized complexity aims to investigate the computational complexity of a problem with respect to a set of interesting parameters, with the goal of understanding if the exponential explosion of an exact algorithm can eventually be confined only to the considered parameters (and not to the overall input). We refer the reader to [68, 80] for an introduction to parameterized complexity.

A preliminary analysis of the parameterized complexity of the One-sided scaffold filling problem started in [77]. The authors presented two Fixed Parameter Tractable (FPT) algorithms for One-sided scaffold filling, under two different parameterizations. In the first case, they considered as parameters the number $k$ of common adjacencies between a filled genome $B'$ and a reference genome $A$, and the maximal number $d$ of occurrences of a gene inside a genome, and gave an FPT algorithm of time complexity $O((2d)^{2k}poly(|A||B|))$. In the second case, the authors considered as parameters the number $k$ of common adjacencies between a filled genome $B'$ and a reference genome $A$ and the size $c$ of the alphabet (that is the set of genes), and gave an FPT algorithm of time complexity $O(c^{2k}poly(|A||B|))$. A natural problem, left open in [77], is to consider the parameterized complexity of One-sided and Two-sided scaffold filling, when parameterized only by the maximum number of common adjacencies $k$.

**Our contribution.** We present two FTP-algorithms for both Scaffold Filling problems, initially presented in preliminary work [7] and then extended in [8], thus answering the open question in [77]. More precisely, we give an algorithm of time complexity $2^{O(k)}poly(|A||B|)$ for One-sided scaffold filling and an algorithm of time complexity $2^{O(k\cdot\log k)}poly(|A||B|)$ for Two-sided scaffold filling, where $k$ is the number of common adjacencies between the resulting genomes ($A$ and $B'$ for the One-sided case, $A'$ and $B'$ for the Two-sided case). We point out that the contribution of [8] is mainly theoretical, since in practice the parameter $k$ is often close to the length of the genomes.

The rest of the chapter is organized as follows. First, in Section 7.2 we introduce some preliminary definitions and we give the formal definition of the two Scaffold Filling

problems. In Section 7.3, we present the FPT algorithm for the One-sided case, while in Section 7.4 we present the FPT algorithm for the Two-sided case.

## 7.2 Preliminaries

Let $\Sigma$ be an *alphabet*, that is a non-empty finite set of symbols. We represent an (unsigned) unichromosomal genome $A$ as a string over alphabet $\Sigma$. It follows that the symbols in $A$ (where each symbol represents a gene) form a multiset on $\Sigma$, denoted by $[A]$. Consider, for example, the string $A = abcdabcdaa$ on alphabet $\Sigma = \{a, b, c, d\}$, then $[A] = \{a, a, a, a, b, b, c, c, d, d\}$. Given a string $A$, we denote by $A[i]$ the symbol of $A$ in $i$-th position, and by $A[i \dots j]$ the substring of $A$ that starts at position $i$ and ends in position $j$. Moreover, we denote the size of $A$ by $|A|$. Note that since we mostly work with multi-sets, operations $\cup$, $\cap$ and $\setminus$ are implicitely understood to be multi-set operations.

Given a string $A$, an *adjacency* of $A$ is an unordered pair of consecutive elements of $A$, that is $A[i]A[i+1]$ or $A[i+1]A[i]$, with $1 \leqslant i \leqslant |A| - 1$. We say that a position $i$, $1 \leqslant i \leqslant |A|$, *induces* an adjacency $ab$, if ($A[i] = a$ and $A[i+1] = b$) or ($A[i] = b$ and $A[i+1] = a$). We denote by $[\![A]\!]$ the multi-set of adjacencies of $A$. Following the previous example, where $A = abcdabcdaa$, we have that the multi-set of adjacencies of $A$ is $[\![A]\!] = \{aa, ab, ab, ad, ad, bc, bc, cd, cd\}$).

The endpoints of $A$, are its first and last position, that is $A[1]$ and $A[|A|]$. In order to deal with endpoints of the two strings, for all the strings we consider, we assume that the first and the last positions contain a dummy symbol $\sharp$, that is not contained in any other position. Formally, for a string $A$, with $|A| = n$, it holds $A[1] = A[n] = \sharp$ and $A[i] \neq \sharp$ when $2 \leqslant i \leqslant n - 1$. Notice that the dummy symbol is not considered when the set of adjacencies $[\![A]\!]$ is defined, i.e. $\sharp A[2]$ and $A[n-1]\sharp$ are not in $[\![A]\!]$.

When comparing two input strings $A$ and $B$, we denote by $X = [A] \setminus [B]$ the multi-set of symbols of $A$ missing in $B$, and by $Y = [B] \setminus [A]$ the multi-set of symbols of $B$ missing in $A$. Given a multi-set of symbols on an alphabet $\Sigma$, a *scaffold* is a string on $\Sigma$ with some missing elements with respect to another string. For the One-sided scaffold filling problem, the multi-set $Y$ is empty.

The two scaffold filling problems we consider in [8] are both based on the definition of *common adjacency* between two genomes (strings) (refer to Fig. 7.1 for an example).

**Definition 7.1.** Consider two strings $A$, $B$ on alphabet $\Sigma$. The multi-set of *common adjacencies* between $A$, $B$ is defined as $[\![A]\!] \cap [\![B]\!]$. A *matching* $M$ of the adjacencies of $A$

and the adjacencies of $B$ is a relation between the positions of $A$ and the positions of $B$ such that:

- for each position $i$ of $A$ or $B$, there exists at most one pair in $M$ containing $i$;
- for each position $i$ of $A$ and $j$ of $B$, $(i, j) \in M$ if and only if position $i$ and position $j$ induce the same adjacency;
- each position that induces a common adjacency belongs to some pair of $M$.

We say that a position of $A$ or $B$ is *matched*, if it belongs to a pair of $M$. Informally, $M$ relates the positions inducing common adjacencies of the two strings $A$ and $B$. Notice that, unlike in permutations, where every position of a permutation inducing a common adjacency matches exactly one position in the other permutation, in strings a position of one string inducing a common adjacency may correspond to many positions of the second string (in Fig. 7.1, notice that position 1 of $A$, inducing adjacency $ab$, can match position 4 or position 5 of $B'$).

Given a scaffold $B$ and a multi-set $X$ of symbols, a string $B'$ is called a *filling* of $B$ with $X$ if

1. $[B'] = [B] \cup X$
2. $B$ is a subsequence of $B'$ such that the first and last symbols of $B'$ are respectively the first and last symbols of $B$.

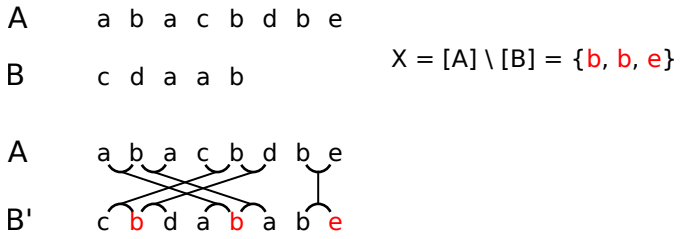



FIGURE 7.1: An instance for the One-sided SF-MNSA problem. Given the complete genome $A$ and the scaffold $B$, we compute the filled genome $B'$ by inserting the symbols of $X$ in $B$. The lines between $A$ and $B'$ connect positions inducing common adjacencies and represent a matching $M$ between the adjacencies of $A$ and $B'$. The number of common adjacencies between $A$ and $B'$ is 5.

Now, we are ready to present the formal definitions of the two Scaffold Filling problems investigated in [8]. Notice that, since we are interested in the parameterized complexity of the two problems, we give the definitions of the parameterized versions of the two problems.

**One-sided Scaffold Filling to Maximize the Number of common String Adjacencies (One-sided SF-MNSA)**

*Input:* Two strings $A$ and $B$, such that $[B] \subseteq [A]$.
*Output:* A filling $B'$ of $B$ with $X = [A] \setminus [B]$ such that $A$ and $B'$ have at least $k$ common adjacencies.
*Parameter:* $k$.

**Two-sided Scaffold Filling to Maximize the Number of common String Adjacencies (Two-sided SF-MNSA)**

*Input:* Two strings $A$ and $B$.
*Output:* A filling $B'$ of $B$ with $X = [A] \setminus [B]$ and a filling $A'$ of $A$ with $Y = [B] \setminus [A]$ such that $A'$ and $B'$ have at least $k$ common adjacencies.
*Parameter:* $k$.

Notice that the One-sided SF-MNSA problem can be seen a restriction of Two-sided SF-MNSA with $Y = \emptyset$.

Now, we discuss some properties that will be useful to design our FPT-algorithms. First, we present the following property for the parameter $k$, proved in [77].

**Lemma 7.2.** *[77] Let $A$ and $B$ two strings, $X = [A] \setminus [B]$, and $Y = [B] \setminus [A]$. Let $k$ be the optimal number of common adjacencies for Two-sided SF-MNSA between two fillings $A'$ and $B'$. Then $|X|, |Y| \leqslant k$.*

Notice that, since One-sided SF-MNSA problem can be seen as a restriction of Two-sided SF-MNSA, Lemma 7.2 holds also for One-sided SF-MNSA, that is when $Y = \emptyset$, it holds $|X| \leqslant k$.

Let $A$ and $B$ be two strings of symbols over an alphabet $\Sigma$, which are input of One-sided SF-MNSA or Two-sided SF-MNSA, and consider the multiset $\mathsf{AD}$ of common adjacencies between $A$ and $B$. We can assume that $|\mathsf{AD}| < k$, otherwise we already know that One-sided SF-MNSA/Two-sided SF-MNSA admits a solution consisting of at least $k$ common adjacencies. Now, since $|\mathsf{AD}| < k$, we can compute a partition of $\mathsf{AD}$ into two subsets as follows:

- the multiset $\mathsf{AD}_{pr} \subseteq \mathsf{AD}$ of common adjacencies that are preserved after the filling of $B$ and/or $A$;
- the multiset $\mathsf{AD}_{br} \subseteq \mathsf{AD}$ of common adjacencies that are broken by inserting symbols of $X = [A] \setminus [B]$ (of $Y = [B] \setminus [A]$ respectively) into $B$ (into $A$ respectively).

Then the following easy property holds.

*Property* 1. Let $A$ and $B$ be two strings on alphabet $\Sigma$ and let $\mathsf{AD}$ be the multiset of common adjacencies between $A$ and $B$. Consider a solution for One-sided SF-MNSA/Two-sided SF-MNSA that partitions $\mathsf{AD}$ into multisets $\mathsf{AD}_{pr}$, $\mathsf{AD}_{br}$, then we can compute the partition of $\mathsf{AD}$ into the two multisets $\mathsf{AD}_{pr}$ and $\mathsf{AD}_{br}$ in time $O(2^k)$.

*Proof.* Recall that $|\mathsf{AD}| < k$, since otherwise we already know that the One-sided/Two-sided SF-MNSA problem admits a solution consisting of at least $k$ common adjacencies. Then, it easy to see that there can be at most $2^k$ subsets $\mathsf{AD}_{pr}$ of $\mathsf{AD}$, hence at most $2^k$ subsets $\mathsf{AD}_{br}$ of $\mathsf{AD}$. □

This property is implicitly used in the two fixed-parameter algorithms, presented in the next sections, in order to guess which adjacencies of the set $\mathsf{AD}$ will be preserved. The latter adjacencies are induced by positions where no insertion is possible when a filling of an input string is computed. Hence, we assume in the following that when a string is inserted into $A$ or $B$, it is not inserted in a position that induces an adjacency in $\mathsf{AD}_{pr}$.

**Color-Coding.** The FPT algorithms we present are based on a well-known technique to design FPT algorithms, called color-coding [81]. Originally introduced to identify subgraphs such as simple paths inside a larger graph [81], color-coding has been applied to many graph problems, for example for the graph motif problem and variant thereof [82–85]. We apply this technique in a different context, that is for string comparison, following some recent examples [86, 87].

Informally, given a set $U$ of size $n$, the color-coding technique aims to find a solution $S \subseteq U$ of size $k$ by coloring the elements of $U$ with $k$ colors, so that each element of $S$ is associated with a distinct color. While enumerating the subsets having size $k$ of $U$ takes time $O(n^k)$, by means of the coloring and applying combinatorial properties of the problem, it is possible in some cases to compute whether a solution of size $k$ exists in time $f(k)poly(n)$, thus leading to an FPT algorithm.

We now introduce the definition of a perfect family of hash functions, which are used to compute the coloring.

**Definition 7.3.** [81] Let $I$ be a set, a family $F$ of hash functions from $I$ to $\{c_1, \ldots, c_k\}$ is called *perfect* if for any subset $I' \subseteq I$, with $|I'| = k$, there exists a function $f \in F$ which is injective on $I'$.

A perfect family $F$ of hash functions from $I$ to $\{c_1, \ldots, c_k\}$, having size $O(\log |I| 2^{O(k)})$, can be constructed in time $O(2^{O(k)}|I| \log |I|)$ [81].

## 7.3 An FPT algorithm for One-sided SF-MNSA

In this section we present an FPT algorithm for the One-sided SF-MNSA problem parameterized by the number $k$ of common adjacencies between the input string $A$ and a filling $B'$ of $B$ with the multi-set $X$. We recall that $X = [A] \setminus [B]$ and that by Lemma 7.2, it holds $|X| \leqslant k$. Furthermore, we assume that we have already computed the two subset $\mathsf{AD}_{pr}$ and $\mathsf{AD}_{br}$ of $\mathsf{AD}$ (the common adjacencies of $A$, $B$) and that no insertion is possible during the filling in a position associated with an adjacency of $\mathsf{AD}_{pr}$ by the matching $M$ of the common adjacencies of $A$ and $B$ (see Prop. 1).

Let $C_A = \{c_1, \ldots, c_k\}$ be a set of *colors*. Consider a family $F$ of perfect hash functions from the positions inducing the adjacencies of $A$ in $\mathsf{AD}_{br}$ (recall that $\mathsf{AD}_{br} = \mathsf{AD} \setminus \mathsf{AD}_{pr}$) to colors in $C_A$. Informally, the coloring is used to identify those positions of $A$ that, due to the insertion of symbols in $X$, match positions of $B$, hence inducing new adjacencies.

In the following, we consider a function $f \in F$, for the positions of $A$, and we assume that such a function $f$ is injective with reference to a filling $B'$ of $B$, that is if there exists a filling $B'$ that induces $k$ common adjacencies, we assume that the positions of $A$ inducing these common adjacencies are colored with $k$ distinct color by $f$.

Given a string $S$, $S$ is said *colorful for* $C_A$ if there exist $\{s_c \mid c \in C_A\} \subseteq (\llbracket S \rrbracket \cap \llbracket A \rrbracket)$, such that for each $c \in C_A$ there is a position of $A$ colored by $c$ which induces the adjacency $s_c$. Our objective is thus to compute a filling $B'$ of $B$ which is colorful for $C_A$.

The algorithm is based on two levels of dynamic programming recurrences. First, we present a dynamic programming recurrence to compute if there exists a string on $X$ to be inserted at a given position $j$ of $A$ so that the resulting string is colorful for a given set $C_j \subseteq C_A$. Then, this result is used to define the dynamic programming recurrence for the insertion of a set of symbols in the string $B[1, j]$ so that the result is colorful for a set $C' \subseteq C_A$.

**Inserting symbols at a specific position.**

We first focus on inserting a given set of symbols at one given position of $B$. Given a position $j$ in $B$, two sets $X_j \subseteq X$ and $C_j \subseteq C_A$, define $\texttt{Ins}_j\,(X_j, C_j) \in \{0, 1\}$ as follows:

$$\texttt{Ins}_j\,(X_j, C_j) = 1 \Leftrightarrow \text{there exists a filling of } B[j-1, j] \text{ with } X_j \text{ which is colorful for } C_j.$$

Note that $\texttt{Ins}_j(\emptyset, \emptyset) = 1$ for all $j$. In the following lemma, we show that $\texttt{Ins}$ can be computed in time $O(2^{2k}k^2)$ by dynamic programming.

**Lemma 7.4.** *We can compute the values of* $\texttt{Ins}_j(X_j, C_j)$ *— where* $X_j \subseteq X$, $C_j \subseteq C_A$ *with* $|X_j|, |C_j| \leqslant k$, *and* $j$ *is an integer with* $1 < j \leqslant |B|$ *— in overall time* $O(2^{2k}k^2|B|)$.

We compute the table $\texttt{Ins}_j(X_j, C_j)$ by dynamic programming. More precisely, for any fixed $j$, we compute a table $\texttt{Add}_\alpha(X', C') \in \{0, 1\}$ defined over all subsets $X' \subseteq X$, $C' \subseteq C$ and symbols $\alpha \in X'$ as follows:

$$\texttt{Add}_\alpha\left(X', C'\right) = 1 \Leftrightarrow \begin{array}{l}\text{there exists a string } s' \text{ with symbols } [s'] = X' \text{ so that} \\ \quad \text{the concatenation } s = B[j-1]s' \text{ is colorful for } C', \\ \quad \text{and the rightmost symbol of } s \text{ is } \alpha\end{array}$$

Intuitively, $\texttt{Add}$ gives, for any set of colors $C'$, the strings formed over $X$ which are colorful for $C'$ if inserted after $B[j-1]$. In fact, rather than computing the precise ordering of each such string, the table only focuses on the set of symbols ($X'$) and the last symbol ($\alpha$) for each one. Moreover, for any given $j$, the table $\texttt{Add}$ contains enough information to deduce the values of $\texttt{Ins}_j(X_j, C_j)$ for all pairs $(X_j, C_j)$.

We prove that table $\texttt{Add}$ can be computed using the following dynamic programming recurrence.

*Recurrence* 1. Let $X' \subseteq X, C' \subseteq C_A$, $\alpha \in \Sigma$.

- if $X' = \emptyset$, then $\texttt{Add}_\alpha(\emptyset, C') = 1$ iff $C' = \emptyset$ and $\alpha = B[j-1]$
- if $|X'| > 0$ and $\alpha \notin X'$, then $\texttt{Add}_\alpha(X', C') = 0$.
- if $|X'| > 0$ and $\alpha \in X'$, then:

$$\texttt{Add}_\alpha\left(X', C'\right) = \max_{\beta \in X' \setminus \{\alpha\}} \begin{cases} \texttt{Add}_\beta(X' \setminus \{\alpha\}, C') \\ \vee\, \exists c \in C', \quad \texttt{Add}_\beta(X' \setminus \{\alpha\}, C' \setminus \{c\}) \text{ and there exists} \\ \qquad\qquad \text{a position of } A \text{ colored by } c \text{ that induces} \\ \qquad\qquad \text{an adjacency } \alpha\beta \end{cases}$$

*Proof.* Base case. Easily, any string $s'$ with $[s'] = \emptyset$ yields $s = B[j-1]s' = B[j-1]$. Hence $s$ is only colorful for the empty set of colors, and its last symbol is $\alpha = B[j-1]$.

Recurrence. The case where $\alpha \notin X'$ is equally trivial (any $s'$ with $[s'] = X'$ would be non-empty and have a last letter in $X'$, and so would $s = B[j-1]s'$). We now consider the case where $\alpha \in X'$. Assume that, for some $\beta \in X' \setminus \{\alpha\}$, we have $\texttt{Add}_\beta(X' \setminus \{\alpha\}, C') = 1$ or $\exists c' \in C'$, $\texttt{Add}_\beta(X' \setminus \{\alpha\}, C' \setminus \{c'\}) = 1$. In the former case there exists a string $s'$ such that $B[j-1]s'$ is colorful for $C'$, hence also $B[j-1]s'\alpha$ is colorful for $C'$. In the latter case there exists a string $s'$ such that $B[j-1]s'$ is colorful for $C' \setminus \{c'\}$, where $\beta$ is the last symbol of $s'$. Moreover, there exists a position of $A$ colored by $c'$ that induces an adjacency $\beta\alpha$, hence string $s = B[j-1]s'\alpha$ is colorful for $C'$. Note that in both cases, $[s'] = X' \setminus \{\alpha\}$ and $[s'\alpha] = X'$.

Reciprocally, assume that $\texttt{Add}_\alpha(X', C') = 1$, i.e. there exist strings $s'$ and $s$ ending with $\alpha$ such that $[s'] = X'$ and $s = B[j-1]s'$ is colorful for $C'$. Let $j = |s|$, and consider the substring $s^* = s[1, j-1]$. Then $s^*$ is colorful for some $C^* \subseteq C'$ and write $\beta$ for the last symbol of $s^*$. By definition, $\texttt{Add}_\beta(X' \setminus \{\alpha\}, C^*) = 1$. Now, we have two possible cases. If $\beta\alpha$ does not induce a common adjacency (that is $s^*$ is colorful for $C'$), then $\texttt{Add}_\beta(X' \setminus \{\alpha\}, C^*) = 1$. If $\beta\alpha$ induces a common adjacency (that is $s^*$ is colorful for $C^* = C' \setminus \{c'\}$, for some $c' \in C'$), then $\texttt{Add}_\alpha(X' \setminus \{\alpha\}', C' \setminus \{c'\}) = 1$. In both cases, the recurrence correctly sets $\texttt{Add}_\alpha(X', C')$ to 1.

□

We now have all the tools to prove Lemma 7.4.

*of Lemma 7.4.* For any $j$, $1 < j \leqslant |B|$, the table $\texttt{Add}$ can be computed using Recurrence 1. It is easy to deduce the values of $\texttt{Ins}_j$:

$$\texttt{Ins}_j(X_j, C_j) = \max_{\alpha \in X_j} \begin{cases} \texttt{Add}_\alpha(X_j, C_j) \\ \vee\, \exists c \in C, \quad \texttt{Add}_\alpha(X_j, C_j \setminus \{c\}) \text{ and there exists} \\ \qquad\qquad \text{a position of } A \text{ colored by } c \text{ that induces} \\ \qquad\qquad \text{an adjacency } \alpha B[j] \end{cases}$$

Indeed, if $\texttt{Ins}_j(X_j, C_j) = 1$, then it follows that there is a substring $s$ over alphabet $X_j$, such that $B[j-1]sB[j]$ is colorful for $C_j$. Write $\alpha$ for the last symbol of $B[j-1]s$. Then, either the substring $B[j-1]s$ is colorful for $C_j$, hence $\texttt{Add}_\alpha(X_j, C_j) = 1$, or $B[j-1]s$ is colorful for $C_j \setminus \{c'\}$, for some color $c' \in C_j$, hence $\texttt{Add}_\alpha(X_j, C_j \setminus \{c'\}) = 1$, and there exists a position of $A$ colored by $c'$ that induces an adjacency $\alpha B[j]$. The reciprocal is clear with a similar decomposition.

Now, we discuss the time complexity of computing $\texttt{Add}_\alpha(X', C')$. Table $\texttt{Add}_\alpha(X', C')$ consists of $O(2^{2k}k)$ entries, since $|X'|, |C'| \leqslant k$, hence we have $2^k$ possible subsets of

each $X' \subseteq X$, $C' \subseteq C_A$. Each entry is computed in time $O(k^2)$, since the algorithm looks for at most $k^2$ entries in the table, depending on the chosen $\beta \in X'$ and $c \in C'$, and $|X'|, |C'| \leqslant k$. Given the table $\texttt{Add}_\alpha(X', C')$, the time complexity to compute each entry $\texttt{Ins}_j(X_j, C_j)$ is $O(k^2)$, since again in the worst case we have to look for $k^2$ entries, depending on the chosen $\alpha \in X_j$ and $c \in C_j$. This process has to be repeated $O(n)$ times, varying $j \leqslant |B|$. Hence the overall time complexity to compute the whole table $\texttt{Ins}_j(X_j, C_j)$ is $O(2^{2k}k^2n)$. □

**Inserting symbols in a prefix of $B$.**

Next we consider the general problem of inserting a multiset of symbol in different positions of a prefix of $B$.

Given a multiset $X' \subseteq X$ of symbols and a set $C'_A \subseteq C_A$ of colors, define $\texttt{Fill}_j(X', C'_A) \in \{0, 1\}$ as follows:

$$\texttt{Fill}_j\left(X', C'_A\right) = 1 \Leftrightarrow \text{there exists a filling of } B[1, \ldots, j] \text{ with } X' \text{ which is colorful for } C'_A.$$

We now prove that $\texttt{Fill}_j(X', C'_A)$ satisfies the following recurrence property. The objective, as stated in Lemma 7.5, is to determine whether a prefix of $B$ can be filled with a given multiset of symbols $X'$ contained in $X$, so that the resulting filling is colorful for a subset of $C_A$.

*Recurrence* 2. Let $X' \subseteq X$, $C'_A \subseteq C_A$.

- For $j = 2$, $\texttt{Fill}_2(X', C'_A) = \texttt{Ins}_2(X', C'_A)$.
- For all $j > 2$,

$$\texttt{Fill}_j\left(X', C'_A\right) = \max_{X_j \subseteq X', C_j \subseteq C'_A} \begin{cases} \texttt{Fill}_{j-1}(X' \setminus X_j, C'_A \setminus C_j) \\ \wedge\, \texttt{Ins}_j(X_j, C_j) \end{cases}$$

*Proof.* Base case. The case $j = 2$ is easily deduced by definition: $\texttt{Fill}_j(X', C'_A) = 1$ if and only if $\texttt{Ins}_j(X', C'_A) = 1$.

Consider $j > 2$, and assume that $\texttt{Fill}_j(X', C'_A) = 1$, that is there exists a filling $B'$ of $B[1, \ldots, j]$ with $X'$ colorful for $C'_A$. Consider the set of symbols $X_j \subseteq X$ inserted in position $j$ of $B$ and let $C_j$ be a set of colors such that there exists a set of positions of $A$

colored by $C_j$ associated by the matching with adjacencies using elements of $X_j$. By definition, $\texttt{Ins}_j(X_j, C_j) = 1$ and $\texttt{Fill}_{j-1}(X' \setminus X_j, C'_A \setminus C_j) = 1$, hence the recurrence formula correctly sets $\texttt{Fill}_j(X, C_A)$ to 1.

Assume now that $\texttt{Ins}_j(X_j, C_j) = 1$ and $\texttt{Fill}_{j-1}(X' \setminus X_j, C'_A \setminus C_j) = 1$ for some $X_j \subseteq X'$, $C_j \subseteq C'_A$, that is the recurrence formula sets $\texttt{Fill}_j(X', C'_A)$ to 1. Then by definition it follows that there exists a filling of $B[1, \ldots, j-1]$ with $X' \setminus X_j$ which is colorful for $C'_A \setminus C_j$ and a filling of $B[j-1, j]$ with $X_j$ which is colorful for $C_j$. Hence concatenating the two fillings (the filling of $B[1, \ldots, j-1]$ and the filling between $B[j-1, j]$), we obtain a filling $B'$ of $B[1, \ldots, j]$ with $X'$ which is colorful for $C'_A$, thus $\texttt{Fill}_j(X', C'_A) = 1$. □

In the following, we prove the correctness of Recurrence 2, that is that $\texttt{Fill}_{|B|}(X, C_A)$ allows us to determine whether $B$ admits a filling with $k$ common adjacencies.

**Lemma 7.5.** *Let $(A, B)$ be an instance of One-sided Scaffold Filling, $X = [A] \setminus [B]$, $k$ be an integer, $C_A$ be a set of $k$ colors, and $F$ be a perfect family of hash functions from the positions of $A$ to $C_A$. Then the following propositions are equivalent:*

*(i) There exists a filling $B'$ of $B$ with $X$ such that $A$ and $B'$ have $k$ common adjacencies;*
*(ii) There exists a coloring $f \in F$ for which $\texttt{Fill}_{|B|}(X, C_A) = 1$.*

*Proof.* (i)⇒(ii) Let $B'$ be a filling of $B$ with $X$ such that $A$ and $B'$ have $k$ common adjacencies; write $I' = [\![A]\!] \cap [\![B']\!]$. Since $|I'| = k$, there exists $f \in F$ such that $f$ is injective on the positions of $A$ that induce $I'$. For each color $c \in C_A$, there exists an adjacency $f(c)$ in $B'$ such that $f(c)$ is induced by a position of $A$ colored by $c$, hence $B'$ is colorful for $C_A$. By definition of $\texttt{Fill}$, we thus have $\texttt{Fill}_{|B|}(X, C_A) = 1$.

(ii)⇒(i) Assume that for some $f \in F$, $\texttt{Fill}_{|B|}(X, C_A) = 1$, then there exists a filling of $B$ with $X$ colorful for $C_A$. For each $c \in C_A$, let $x_c$ be the unique position of $A$ colored by $c$ inducing a common adjacencies with $B'$. Then $\{x_c \mid c \in C_A\}$ is a set of positions of size $k$ yielding $k$ common adjacencies between $A$ and $B'$. □

Next, we show how the recurrence described in Recurrence 2 yields a dynamic programming algorithm to solve One-sided SF-MNSA.

**Theorem 7.6.** *Let $A$, $B$ be two strings of symbols on an alphabet $\Sigma$ and let $X = [A] \setminus [B]$ be the multiset of symbols missing in $B$. It is possible to compute a solution of One-sided SF-MNSA in time $2^{O(k)}poly(|A| + |B|)$.*

*Proof.* By Lemma 7.5, it suffices to compute a family $F$ of hash functions from the positions of $A$ to $C_A$. Then the problem admits a solution if and only if there exists a

coloring $f \in F$ for which $\texttt{Fill}_{|B|}(X, C_A) = 1$. Recurrence 2 together with Lemma 7.4 yields a dynamic programming algorithm to compute $\texttt{Fill}_{|B|}(X, C_A)$ for each coloring.

Now, we consider the time complexity of the algorithm. Write $n = |A| + |B|$. First, a perfect family $F$ of hash functions that color-codes the positions of $A$ can be computed in time $2^{O(k)}poly(n)$. Once the family is computed, the algorithm iterates over the $2^{O(k)}\log(n)$ possible functions $f \in F$ and the respective color codings. For each function $f \in F$, the table $\texttt{Ins}_j(X', C'_A)$ is computed in time $O(2^{2k}k^2|B|)$ (see Lemma 7.4). Then the $O(2^{2k}|B|)$ entries of table $\texttt{Fill}_j(X', C'_A)$ are computed (Recurrence 7.4), where each entry requires $O(2^{2k})$ look-ups, depending on the choice of $X_j$ and $C_j$. Thus the algorithm requires $O(2^{4k}n)$ time to compute table $\texttt{Fill}_j(X', C'_A)$. Finally, the overall complexity is indeed $2^{O(k)}poly(n)$. □

## 7.4 An FPT algorithm for Two-sided SF-MNSA

In this section, we consider the Two-sided SF-MNSA problem and we present a fixed-parameter tractable algorithm for it. As for the One-sided case, the algorithm is based on color-coding and dynamic programming. However, the same approach described in the previous section cannot be applied directly and new challenges make Two-sided SF-MNSA more complicated than One-sided SF-MNSA. First, there exist a new kind of common adjacencies, namely adjacencies that are created in the fillings although they never appear as such in any of the input strings. Also, unlike the One-sided case, it is not known a priori whether a given adjacency of the string $A$ may be used in a common adjacency or should be split to insert a substring. We deal with the first issue by bounding (and enumerating) the possible arrangements of such adjacencies, and with the second by introducing "insertion" colors, so that the positions associated with such colors can only be used to insert a substring and not (directly) to create a common adjacency.

Given two strings $A$ and $B$ over alphabet $\Sigma$, denote by $k$ the number of common adjacencies between two fillings $A'$ and $B'$ of $A$, $B$ respectively. We denote by $X = [A]\backslash[B]$ the multi-set of symbols of $A$ missing in $B$ and by $Y = [B] \setminus [A]$ the multi-set of symbols of $B$ missing in $A$, where $X, Y \neq \emptyset$ (otherwise the problem is equivalent to One-sided SF-MNSA) and $X \bigcap Y = \emptyset$ (by the definition of sets $X$ and $Y$ for the Two-sided SF-MNSA).

Recall that, by Lemma 7.2, the following property holds: $|X|, |Y| \leqslant k$. Furthermore, as in the previous section, we assume that we have already computed the subset $\mathsf{AD}_{pr}$ of

$[\![A]\!] \cap [\![B]\!]$, that is those common adjacencies of $A$ and $B$, that must be preserved during the filling (see Prop. 1).

Before giving the details of the FPT-algorithm, we present an informal overview. A filling $B'$ ($A'$ respectively) of $B$ (of $A$ respectively) consists of inserting substrings on alphabet $X$ (on alphabet $Y$ respectively) into $B$ (into $A$ respectively). Now, the algorithm consists of four steps.

**Step 1**. In the first step, the algorithm "guesses" (that is iterates over all possible cases) how the letters from $X$ and $Y$ should be arranged into strings to be inserted into $A$ and $B$. Such strings are called *filler strings*. Note that we do not guess the insertion point of those strings, and since $|X|, |Y| \leqslant k$, the number of cases to try depends only on a function of $k$ (see Prop. 2).

**Step 2**. The second phase identifies two kinds of common adjacencies for two fillings $A'$, $B'$. In the first kind, one adjacency appears already in $[\![A]\!]$ or $[\![B]\!]$: this case can be dealt with as in the One-sided algorithm. In the second kind, both adjacencies inducing a common adjacency of $A'$ and $B'$ have been created during the filling, using one element from $X$ in $B'$ and one from $Y$ in $A'$. They are called $(X, Y)$-*adjacencies*. Since $X \cap Y = \emptyset$, such adjacencies use *exactly* one element of $X$ and one element of $Y$. Therefore these adjacencies consist of an endpoint of an inserted string as well as a symbol already present in the original strings $A$ and $B$. The second step of the algorithm identifies and matches the endpoints of inserted strings (computed in Step 1) which correspond to such $(X, Y)$-adjacencies (see Def. 7.8 and Prop. 3).

**Step 3**. In the third step, the algorithm opportunely color-codes the positions of $A$ and $B$ with two disjoint sets of colors, in order to:

1. match non $(X, Y)$-adjacencies (like in the previous algorithm)
2. identify the positions of $A$ and $B$ where an insertion is possible (we will show that the number of these positions is bounded by $k$ in Remark 2)

**Step 4**. Step 4 finally inserts the strings into $A$ and $B$ by dynamic programming, while creating the remaining adjacencies (see Recurrence 3).

Now we are able to present the details of the different steps of the algorithm.

**Step 1:** *Compute filler strings.*

Consider a solution of Two-sided SF-MNSA $(A', B')$ inducing $k$ common adjacencies. We call *filler string* a non-empty string consisting of elements of $X$ or of $Y$ inserted into

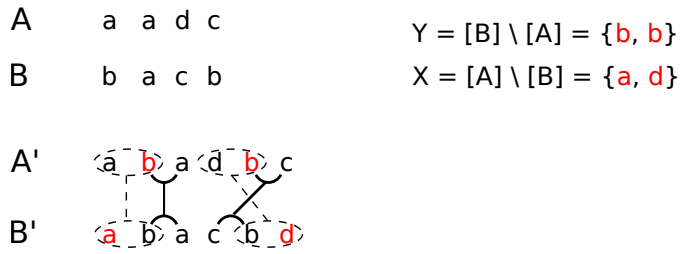



FIGURE 7.2: An instance of the Two-sided SF-MNSA problem. Given two scaffolds $A$ and $B$, we obtain the filled genomes $A'$ and $B'$ by inserting symbols $X$ in $B$ and $Y$ in $A$ (inserted symbols are in red). Straight Lines connect common adjacencies, dotted lines connect $(X, Y)$-adjacencies. Notice for example that the $bd$ common adjacency is an $(X, Y)$-adjacency induced by the insertion of $b$ into string $A$ and of $d$ into string $B$.

$B$ or $A$ to create $B'$ or $A'$. We write $S_X$ and $S_Y$ for the two multi-sets of filler strings over the multi-sets $X$ and $Y$ that are inserted into $B$ and $A$ respectively. The algorithm simply iterates through all such pairs $(S_X, S_Y)$ of multi-sets of strings over $(X, Y)$: in some iteration, the correct pair $(S_X, S_Y)$ will eventually be considered. The following property bounds both the number of possible pairs $(S_X, S_Y)$ and the number of positions where filler strings can be inserted into $A$ and $B$.

*Property* 2. Let $X$, $Y$ be two multi-sets of symbols to be inserted into the string $B$ and $A$ respectively. Then (1) the number of positions in each of $A$, $B$ where a filler string is inserted is bounded by $k$ and (2) the number of possible multi-sets $S_X$ and $S_Y$ of filler strings over $X$, $Y$ to be inserted into $B$ and $A$ respectively is bounded by $O(k^{2k})$.

*Proof.* (1) The property follows easily from the fact that, since $|X|, |Y| \leqslant k$, $|S_X|, |S_Y| \leqslant k$.

(2) By Lemma 7.2, $|X|, |Y| \leqslant k$. Now, consider w.l.o.g. a multi-set $S_X$ of filler strings inserted into $B$. This multi-set obviously consists of at most $k$ strings, where each one has length bounded by $|X| \leqslant k$. Hence, the number of possible multi-sets of filler strings to be inserted into $B$ is bounded by $O(k^k)$. We can conclude that the overall possible multi-sets of filler strings over $X$, $Y$ inserted into $B$ and $A$ respectively, in order to obtain two fillings $B'$, $A'$, is bounded by $O(k^{2k})$. □

**Step 2:** *Identify* $(X, Y)$*-adjacencies.*

We first define formally the concept of $(X, Y)$-*adjacency* (see Fig. 7.2 for an example).

**Definition 7.7.** Consider a filling $B'$ of $B$ with $X$ and a filling $A'$ of $A$ with $Y$. A common adjacency $z \in [\![A']\!] \cup [\![B']\!]$ is an $(X, Y)$-*adjacency* if it is induced by positions $i$, $j$ of $A'$, $B'$ respectively, such that

- one of $A'[i]$ or $A'[i+1]$ is the endpoint of a filler string $s_y \in S_Y$,

- and one of $B'[j]$ or $B'[j+1]$ is the endpoint of a filler string $s_x \in S_X$

Notice that, since $X \cap Y = \emptyset$, it follows that any new common adjacency of $A'$ (of $B'$ respectively) is either involved in only one insertion (hence, in one input string, it is induced by a position where no insertion occurs), or it is an $(X, Y)$-adjacency.

Now, the algorithm considers the endpoints of the filler strings computed in the previous step, and defines which endpoints induce a common $(X, Y)$-adjacency. Denote by $E_X$ ($E_Y$ respectively), the set of endpoints of the strings in set $S_X$ (in set $S_Y$ respectively). Note that we consider that each string yields two end-points, even length-one filler strings. In order to compute which endpoints of $E_X$ and $E_Y$ induce a common $(X, Y)$-adjacency, we use a procedure, called *number assignment*, that associates with each endpoint in $E_X$ and $E_Y$ a number which identifies the $(X, Y)$-adjacency, if any, which uses this endpoint. The procedure assumes that $k'$ is the number of induced $(X, Y)$-adjacencies.

**Definition 7.8.** A *number assignment* for the strings in $S_X \cup S_Y$ is a function from $E_X \cup E_Y$ to $\{0, 1, \ldots, k'\}$, where each number $\{1, \ldots, k'\}$ is assigned to exactly one endpoint in $E_X$ and one endpoint in $E_Y$.

Consider a solution of Two-sided SF-MNSA, a corresponding number assignment is obtained as follows (recall that $k'$ denotes the number of $(X, Y)$-adjacencies). Consider an endpoint $e_z \in E_X \cup E_Y$, then:

- endpoint $e_z$ is associated with 0 if and only if it is not involved in an $(X, Y)$-adjacency;
- endpoint $e_z$ is associated with a number $i \in \{1, \ldots, k'\}$ if and only if it is involved in the $i$-th $(X, Y)$-adjacency.

The set $E'_X \subseteq E_X$ ($E'_Y \subseteq E_Y$) denotes the set of endpoints of $E_X$ (of $E_Y$ respectively) associated with a positive number.

Next, we show how to compute a number assignment in time $O((4k)^{k+1})$. The following property gives an easy upper bound on the number of such assignments.

*Property* 3. There exist at most $(4k)^{k+1}$ number assignments.

*Proof.* Notice that $|E_X \cup E_Y| \leqslant 4k$, since $|S_X|, |S_Y| \leqslant k$. Moreover, each endpoint is assigned a number in $\{0, 1, \ldots, k'\}$, with $k' \leqslant k$, hence there exist at most $(4k)^{k+1}$ number assignments. □

The algorithm iterates over each possible number assignment, hence, in what follows we assume that the algorithm guesses the correct number assignment to $E_X \cup E_Y$.

Once we have defined through the number assignment which endpoints in $E'_X \cup E'_Y$ induce a common adjacency, we have to bound the possible symbols adjacent to an endpoint in $E'_X \cup E'_Y$. Indeed, when we will insert into $A$ a filler string $s_y$ whose endpoint induces an $(X, Y)$-adjacency, we will not be able to define a matching between this adjacency and an adjacency of $B$, because we still have to insert the "companion" strings in $B$ (that is the filler string $s_x$ that induces an $(X, Y)$-adjacency with $s_y$). However, by restricting the possible symbols that must be made adjacent to the filler strings, we will be able to ensure that a common adjacency is eventually induced.

Now, we show how we can restrict the possible symbols that are adjacent to an endpoint in $E'_X \cup E'_Y$. First, we introduce the following definition.

**Definition 7.9.** Consider a solution $(A', B')$ to the scaffold-filling problem and a filler string $s_x \in S_X$ ($s_y \in S_Y$ respectively). Let $e_x \in E'_X$ ($e_y \in E'_Y$ respectively) be an endpoint of $s_x$ (of $s_y$ respectively) yielding an $(X, Y)$-adjacency. Then we define $v(e_x)$ ($v(e_y)$ respectively) as the symbol of $Y$ (of $X$ respectively) adjacent to $e_x$ in $B'$ (to $e_y$ in $A'$ respectively).

The symbols $v(e_x)$, $v(e_y)$, for each $e_x \in E'_X$, $e_y \in E'_Y$ are immediately deduced from the number assignment. Indeed, if $e_x \in E'_X$ and $e_y \in E'_Y$ are associated with the same number $i$ (that is they induce an $(X, Y)$-adjacency), then $v(e_x)$ must be the symbol of the filler strings at endpoint $e_y$, while $v(e_y)$ must be the symbol at endpoint $e_x$.

*Remark* 1. A number assignment uniquely determines the value $v(e_z)$ for $e_z \in E'_X \cup E'_Y$.

*Proof.* Consider the case that $e_x \in E_X$ and $e_y \in E_Y$ are associated by the number assignment with the same number $i \in \{1, \ldots, k'\}$. Since $X \cap Y = \emptyset$, it follows that $e_x$ must be inserted in a position of $B$ containing the same symbol as $e_y$ and $e_y$ must be inserted in a position of $A$ containing the same symbol as $e_x$. □

Using the number assignment and the values $v(e_z)$, with $e_z \in E'_X \cup E'_Y$, the algorithm creates the following table which tells whether, according to $(X, Y)$-adjacencies, a filler string can be inserted at a certain position. We define the table for filler strings of $S_X$, the definition for $S_Y$ being similar. Let $j \in \{1, \ldots, |B|\}$, $s$ be a filler string in $S_X$, and $s_l, s_r$ for the left and right endpoints of $s$ respectively:

$$\texttt{XY-Fits}_{B,j}(s) = \begin{cases} 0 & \text{if } (s_l \in E'_X \text{ and } B[j-1] \neq v(s_l)) \\ & \text{or } (s_r \in E'_X \text{ and } B[j] \neq v(s_r)) \\ 1 & \text{otherwise.} \end{cases}$$

**Step 3:** *Color-code the positions in $A$ and $B$.*

Our next goal is to distinguish, for each input string, adjacencies that need to be broken to insert a filler string, from adjacencies that will yield a common adjacency when the other filling is created. Since there are at most $k$ adjacencies of either kind, we use color-coding to achieve this goal. Consider a coloring $f \in F$ of the positions of $A$ and $B$ with a set $C$ of $z$, $k \leqslant z \leqslant 2k$, colors. We partition $C$ into disjoint subsets $C_{M,A}$, $C_{M,B}$, $C_{I,A}$, $C_{I,B}$ defined as follows:

- Let $C_{M,B}$ ($C_{M,A}$ respectively) be a set of colors associated with positions of $B$ (of $A$ respectively) that matches positions of $A'$ (of $B'$ respectively). Notice that in a position colored by $C_{M,A}$ ($C_{M,B}$ respectively) a string of $S_X$ (of $S_Y$ respectively) cannot be inserted.
- Let $C_{I,B}$ ($C_{I,A}$ respectively) be a set of colors assigned to positions in $B$ (in $A$ respectively) where insertions of strings of $S_X$ (of $S_Y$ respectively) are allowed.

Since the two multisets of strings $S_X$, $S_Y$ are fixed, and the number assignment of the $k'$ $(X,Y)$-adjacencies is fixed, we only consider partitions where $|C_{I,A}| = |S_Y|$, $|C_{I,B}| = |S_X|$, and $|C_{M,A}| + |C_{M,B}| + k' = k$.

**Step 4:** *Insert strings by dynamic programming.*

Now we have all the tools to decide where each string must be inserted. Thanks to Steps 1–3, we can now deal with both sides independently, hence without loss of generality we describe the algorithm inserting filler strings of $S_X$ in $B$. The constraints one needs to observe are the following.

- Filler strings are inserted at positions colored by $C_{I,B}$. Note that we do not require to insert a filler string in *every* colors of $C_{I,B}$: we only need to ensure that no adjacency having a color in $C_{M,B}$ is broken.
- $(X,Y)$-adjacencies are created as guessed during Step 3, that is each filler string $s$ inserted at position $j$ yields $\texttt{XY-Fits}_{B,j}(s) = 1$

- The remaining created adjacencies yield enough common adjacencies with $A$. More precisely, for each color $c \in C_{M,A}$, we create at least one adjacency which can be matched to an adjacency with color $c$ in $A$.

The first two constraints can clearly be checked in constant time for any filler string $s$ at any position $j$. The third constraint is dealt with as follows.

Let $s \in S_X$ be a filler string, and $j$ be a position of $B$. Let $H$ be the substring of $B[j-1]sB[j]$ from which $B[j-1]$ is removed if $s_l \in E'_B$ and from which $B[j]$ is removed if $s_r \in E'_B$. That is, if the string $s$ is inserted at position $j$ in $B$, then $H$ covers the positions which may create a common adjacency which is not an $(X,Y)$-adjacency. In order to determine whether $H$ induces enough new common adjacencies for a given set of colors, we define $\texttt{Col-Fits}_{B,j}(s, C_j) \in \{0,1\}$ for all $C_j \subseteq C_{M,A}$:

$$\texttt{Col-Fits}_{B,j}(s, C_j) = 1 \Leftrightarrow H \text{ is colorful for } C_j$$

Combining with the first two constraints, and similarly to the One-sided case, we define $\texttt{Ins}_{B,j}(s, C_j) \in \{0,1\}$, where $C_j \subseteq C_{M,A}$, as follows:

$$\begin{aligned} \texttt{Ins}_{B,j}(s, C_j) = 1 \Leftrightarrow \quad & B[j-1] \text{ is assigned a color in } C_{I,B} \\ & \wedge\, \texttt{XY-Fits}_{B,j}(s) = 1 \\ & \wedge\, \texttt{Col-Fits}_{B,j}(s, C_j) = 1 \end{aligned}$$

We extend the definition to deal with the empty string $\varepsilon$:

$$\texttt{Ins}_{B,j}(\varepsilon, C_j) = 1 \Leftrightarrow\ C_j = \emptyset$$

Here table $\texttt{Ins}_{B,j}(s, C_j)$ is easier to compute than in the one-sided case, because the string to be insterted is known (and not only the set of its elements). Each entry can be computed in time $O(k^3)$ using a matching algorithm.

**Lemma 7.10.** *Let $j$ be an integer s.t. $j \leqslant |B|$ and $C_j \subseteq C_{M,B}$. Then we can compute $\texttt{Ins}_{B,j}(s, C_j)$ in time $O(k^3)$.*

*Proof.* Recall that the color of $B[j-1]$ is known from Step 3 and that $\texttt{XY-Fits}_{B,j}(s)$ is directly deduced from the number assignment computed at Step 2. Hence computing $\texttt{Ins}_{B,j}(s, C_j)$ only requires to compute the value of $\texttt{Col-Fits}_{B,j}(s, C_j)$, which in turn goes down to deciding whether a given string $H$ is colorful for a given set of colors $C_j$. This can be achieved as follows.

Let $\mathbb{H} = [\![H]\!]$ be the multiset of adjacencies of $H$ (that is, the ones created by inserting string $s$ which need to cover all colors in $C_j$). Create a bipartite graph $G$ with vertex set $\mathbb{H} \cup C_j$. Add an edge $(h, c)$ for each $h \in \mathbb{H}$ and $c \in C_j$ such that there exists a position with color $c$ inducing an adjacency $h$. Then string $H$ is colorful for $C$ if and only if graph $G$ admits a matching covering all vertices of $C_j$. The existence of such a matching can be determined in time $O(\sqrt{|\mathbb{H}| + |C_j|}|\mathbb{H}||C_j|) = O(k^3)$ [88]. We can then deduce the values of $\texttt{Col-Fits}_{B,j}(s, C_j)$ and $\texttt{Ins}_{B,j}(s, C_j)$. □

We can now compute a filling of $B$ satisfying all the above constraints. We do this by dynamic programming, i.e. filling progessively $B$ while keeping track of inserted substrings and covered colors.

**Definition 7.11.** Let $S'_X \subseteq S_X$, $C'_{M,A} \subseteq C_{M,A}$, and $1 \leqslant j \leqslant |B|$. We define $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) \in \{0, 1\}$ as follows. $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ if and only if there exists a filling $B'$ of $B[1, \ldots, j]$ such that:

1. $B'$ is obtained by inserting all strings in $S'_X$ in different positions of $B[1, \ldots, j]$,
2. the inserted strings are inserted at positions colored by $C_{I,B}$,
3. for any inserted string $s \in S_X$ at position $j$, $\texttt{XY-Fits}_{B,j}(s) = 1$,
4. the filling is colorful for $C'_{M,A}$ after removing $(X, Y)$-adjacencies.

We observe that the entries of `Fill-B` can be computed using the following dynamic programming recurrence.

*Recurrence* 3. Let $S'_X \subseteq S_X$, $C'_A \subseteq C_A$.

- For $j = 1$, $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ iff $S'_X = \emptyset$ and $C'_{M,A} = \emptyset$.
- For all $j \geqslant 2$,

$$\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = \max_{\substack{s_j \in S_X \cup \{\varepsilon\} \\ C_j \subseteq C'_{M,A}}} \begin{cases} \texttt{Fill-B}_{j-1}\left(S'_X \setminus \{s_j\}, C'_{M,A} \setminus C_j\right) \\ \wedge\, \texttt{Ins}_{B,j}(s_j, C_j) \end{cases}$$

*Proof.* In case $j = 1$, note that no substring can be inserted in the size-1 string $B[1 \ldots 1]$, hence $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ implies that $S'_X = \emptyset$. Note that in this case ($S'_X = \emptyset$), no common adjacencies can be created, hence $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ if and only if $C'_{M,A} = \emptyset$.

We now prove the recurrence formula.

Assume that $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$. Let $B'$ be a filling of $B[1,\ldots,j]$ using $S'_X$ and colorful for $C'_{M,A}$. If there is no string inserted at position $j-1$ (i.e. in $B[j-1,j]$), then we directly have $\texttt{Fill-B}_{j-1}\left(S'_X, C'_{M,A}\right) = 1$. Using $s_j = \varepsilon$, and since $\texttt{Ins}_{\emptyset}\,(B,\varepsilon)\,j = 1$, the formula is correct in this case. Otherwise, assume that a string $s_j \in S_X$ is included between $B[j-1]$ and $B[j]$, creating a string $B[j-1]s_jB[j]$ that is colorful for some $C_j \subseteq C'_{M,A}$ (after removing $(X,Y)$-adjacencies). Then, for this particular string $s_j$ and this subset $C_j$, we have both $\texttt{Fill-B}_{j-1}\left(S'_X \setminus \{s_j\}, C'_{M,A} \setminus C_j\right) = 1$ and $\texttt{Ins}_{B,j}\,(s_j, C_j) = 1$. Thus, again, the recurrence formula correctly sets $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right)$ to 1.

Conversely, assume that

$$\max_{\substack{s_j \in S_X \cup \{\varepsilon\} \\ C_j \subseteq C'_{M,A}}} \begin{cases} \texttt{Fill-B}_{j-1}\left(S'_X \setminus \{s_j\}, C'_{M,A} \setminus C_j\right) \\ \wedge\, \texttt{Ins}_{C_j}\,(B, s_j)\,j \end{cases} = 1$$

Consider first the case where the maximum is obtained for $s_j = \varepsilon$ and some $C_j \subseteq C'_{M,A}$. Then $\texttt{Ins}_{C_j}\,(B,\varepsilon)\,j = 1$ yields $C_j = \emptyset$. Hence $\texttt{Fill-B}_{j-1}\left(S'_X, C'_{M,A}\right) = 1$, and there exists a filling of $B[1,\ldots,j-1]$ which is trivially extended to a filling $B'$ of $B[1,\ldots,j]$ by adding element $B[j]$. Filling $B'$ directly satisfies Definition 7.11. Otherwise, the max is obtained for some $s_j \in S'_X$ and some $C_j \subseteq C'_{M,A}$. We thus obtain a filling of $B[1,\ldots,j-1]$, and augment this filling by adding string $s_jB[j]$. It is easy to check that this new filling satisfies Definition 7.11. In both cases, we have $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ □

The following Lemma sums up all the results from Steps 1–4. Note that since Step 4 must be run for both $A$ and $B$ independently, we define and compute a table `Fill-A` similarly to table `Fill-B`.

**Lemma 7.12.** *Let $(A,B)$ be an instance of Two-sided Scaffold Filling, $X = [A] \setminus [B]$, $Y = [B] \setminus [A]$, $k$ be an integer, $C$ be a set of colors, and $F$ be a perfect family of hash functions from the positions of $A$ and $B$ to $C$. Then the following propositions are equivalent:*

*(i) There exists a filling $A'$ of $A$ with $Y$ and a filling $B'$ of $B$ with $X$ such that $A'$ and $B'$ have $k$ common adjacencies;*
*(ii) There exist two multi-sets of strings $S_X$ and $S_Y$ over $X$, $Y$, a number assignment, a color-coding $f \in F$ and a partition $C = C_{M,A} \cup C_{M,B} \cup C_{I,A} \cup C_{I,B}$ such that $\texttt{Fill-A}_{|A|}\,(S_Y, C_{M,B}) = \texttt{Fill-B}_{|B|}\,(S_X, C_{M,A}) = 1$.*

*Proof.* (i)⇒(ii) Let $A'$ be a filling of $A$ with $Y$ and $B'$ be a filling of $B$ with $X$ such that $A'$ and $B'$ have $k$ common adjacencies;

Define $S_X$, $S_Y$, a number assignment, a color-coding $f \in F$ and a partition $C = C_{M,A} \cup C_{M,B} \cup C_{I,A} \cup C_{I,B}$ according to these two fillings, so that Definition 7.11 is satisfied. Then it follows that $\texttt{Fill-A}_{|A|}(S_Y, C_{M,B}) = \texttt{Fill-B}_{|B|}(S_X, C_{M,A}) = 1$.

(ii)⇒(i) Reciprocally, assume that $\texttt{Fill-A}_{|A|}(S_Y, C_{M,B}) = 1$ and $\texttt{Fill-B}_{|B|}(S_X, C_{M,A}) = 1$. Then it follows that $S_X$, $C_{M,A}$, $C_{I,B}$ satisfy Definition 7.11 for $B[1, |B|]$. The same property holds for $S_Y$, $C_{M,B}$, $C_{I,A}$ satisfying Definition 7.11 for $A[1, |A|]$. This leads to a filling of $A$ and $B$ with $k$ adjacencies as follows: $k'$ $(X, Y)$-adjacencies obtained from the endpoints of the corresponding inserted strings (remember that $\texttt{XY-Fits}_{B,j}(s) = 1$ for any $s$ inserted at a position $j$ in string $B$), and at least $|C_{M,A}|$ (resp $|C_{M,B}|$) other common adjacencies between the filling of $B$ (resp. the filling of $A$) and $A$ (resp. of $B$). Note that these last common adjacencies appear also in the filling of $A$ (resp., of $B$) since no substring may break them (all substring are inserted in positions colored by $C_{I,B}$ and $C_{I,A}$). Thus there are, overall, $k' + |C_{M,A}| + |C_{M,B}| = k$ common adjacencies between the fillings of $A$ and $B$. □

We present now the main result of this section.

**Theorem 7.13.** *Let $A$, $B$ be two strings over alphabet $\Sigma$ and let $X = [A] \setminus [B]$ be the multiset of symbols of $A$ missing in $B$ and $Y = [B] \setminus [A]$ the multiset of symbols of $B$ missing in $A$. It is possible to compute a solution of Two-sided SF-MNSA over instance $(A, B)$ in time $2^{O(k \log k)} poly(n)$.*

*Proof.* The correctness of the algorithm is directly given by Lemma 7.12: once a perfect family of hash functions $F$ is fixed and two multi-sets of strings $S_X$ and $S_Y$ over $X$, $Y$, a number assignment, a color-coding $f \in F$ and a partition $C = C_{M,A} \cup C_{M,B} \cup C_{I,A} \cup C_{I,B}$ are selected by exhaustive branching, it suffices to compute the entries $\texttt{Fill-A}_{|A|}(S_Y, C_{M,B})$ and $\texttt{Fill-B}_{|B|}(S_X, C_{M,A})$, and return the corresponding fillings of $A$ and $B$ if both entries are equal to 1.

The time complexity of the algorithm is dominated by the iteration over all possible pairs $(S_X, S_Y)$ and of the number assignment. The number of possible sets $S_X$, $S_Y$ is bounded by $k^{2k}$ from Prop. 2. By Prop. 3 there are $O(4k^{k+1})$ number assignments to iterate through. The dynamic programming recurrence requires time $O(2^{4k}n)$.

Consider now, the color-coding. There exists $k$ values of $z$ to test. For each $z$, there are $O(2^{O(z)} \log n)$ colorings [81], and for each coloring, $4^z$ ways of partitionning $C$ into $C_{M,A}$, $C_{M,B}$, $C_{I,A}$, $C_{I,B}$. Overall, there are thus $O(2^{O(k)} \log n)$ cases to consider.

Since a family of perfect hash function of size $O(2^{O(k)} poly(n))$ can be computed in time $O(2^{O(k)} poly(n))$ [81], and the possible partitions of $C$ into sets $C_{M,A}$, $C_{M,B}$, $C_{I,A}$, $C_{M,B}$

are less than $2^{4k}$ (including the constraint $|C_{M,A}| + |C_{M,B}| + k' = k$), it follows that the overall time complexity of the algorithm is bounded by $O((2k)^{2k+1}2^{O(k)}poly(n)) = 2^{O(k \log k)}poly(n)$. □

# Chapter 8

# Conclusions and future research

This dissertation explores two main research directions sharing the common goal of studying the evolutionary history of populations or genomic information. The first research direction investigated regards the development of simulation algorithms for sampling ancestral recombination graphs representing the evolution of multiple present day populations of a single species that are related and admixed. On the other hand, our effort is also devoted to the development of combinatorial algorithms for the reconstruction of phylogenesis representing the evolutionary history of several species or taxa (described by a set of attributes or characters). Part of the thesis is also dedicated to deal with the problem of completing draft incomplete genomic sequences, called scaffolds, produced by NGS technologies and used in comparative and reconstruction analyses.

In the following, for each problem we faced, we summarize our main contributions and we point out some future research directions and open problems we consider worthwhile and interesting.

## 8.1 SimRA: Sampling ARG of multiple populations - Ch. 3

The literature on reconstruction of ancestral recombination graphs (ARGs) has advanced considerably in recent years, mostly due to the fact that genetic variation data is now available on large enough scales to have hope of reconstructing them. However, method developers generally need simulated data because real population genetic data rarely come with a known ARG. Hence, it is valuable to this community to have good tools for rapidly simulating large numbers of ARGs. Consequently, simulating complex evolution scenarios of multiple populations is an important task in population genomics. Apart from the population samples, obtained by simulation, the underlying ARG is an additional important vehicle in checking hypothesis and reconstruction studies.

With this aim in mind, in [1] we present an algorithm SimRA that simulates generic multiple population evolution model with admixture. SimRA is based on random graphs and improves dramatically, in terms of time and space requirements, the classical backward simulation algorithm for single population (Hudson algorithm [10, 26]) . Our experimental analysis reveals that SimRA performs better specially with higher values of recombination rate $r$, although both are backward simulators (see Section 3.6). Additionally, SimRA also simulates STR polymorphisms. In this framework, the ARG is a random graph defined on a set of parameters (i.e., effective population size, sample size, recombination rate, segment length and mutation rate). Moreover, complex scenarios simulations of multiple population evolution require an excess of interdependent parameters making even the scenario-specification highly nontrivial. Using the underlying random graphs model, we derive closed form functions of expected values of the ARG characteristics in terms of its defining parameters (see Section 3.9). These derivations use the graph-theoretic results of the random graph model presented in [19]. More precisely, we consider the following hallmarks, or characteristics, of an ARG: height of the graph, number of recombinations, number of mutations and population diversity. To the best of our knowledge this is the first time closed form expressions have been computed for the ARG properties. What is the need for closed-form values of these characteristics ? Apart from an effort at mathematical completeness, firstly they provide a framework for checking the correctness of ARG sampling algorithms and secondly they represent an useful tool in aiding the user to specify meaningful parameters for complex population scaffold architectures. In other words, the closed form functions are crucial in helping the user to design parameter set-up for complex scenario simulations, not through trial-and-error based on raw computational power but intelligent parameter estimation, thus removing time consuming trials and error iterations.

Moreover, we give an example of use of the expected height of a truncated ARG for designing complex scenarios of multiple population evolution. Indeed, we may ask what should be the sample size $m$ of a population such that the expected number of active lineages in $t$ generations is more than one. See the case study presented in Section 3.11. We show, through simulations, that the expected values closely match the empirical values.

Through comparison studies, we demonstrate that the ARGs produced by SimRA are more compact (i.e., the redundancies are drastically reduced in SimRA), without compromising any accuracy. Indeed, extensive experiments illustrate that the accuracy of the two algorithms (SimRA and Hudson) are comparable (see Section 3.10), while SimRA outperforms Hudson in time, space and non-redundancy factor.

SimRA is both time and space efficient enough to be practical, indeed it makes possible to run hundreds of experiments in very short time (in minutes) enabling a very effective vehicle of carrying out complex studies, such as in [3].

A relevant future direction is to incorporate natural selection in the single population mode of SimRA, since natural selection is what drives evolution. SimRA is based on the neutral Write-Fisher model of evolving population, but it is challenging to model selection in our backward simulation algorithm in order to consider its effects on the resulting topology and samples.

## 8.2 Topological Signatures for Population Admixture - Ch. 4

In [3] the first combinatorial approach to characterize admixture in populations, based on ARGs, has been presented. Traditionally admixture has been addressed by studying linkage disequilibrium distributions. Through controlled simulations, we show that it is feasible to detect admixture by topological signatures.

Moreover, when the model is applied on avocado germplasm data, we observe similar signatures of the persistent cycles, as is seen in the simulation experiments. Due to noise and other unknown factors in real data, the signatures may require to be calibrated (i.e., values of $c$ in Section 4.3) based on training data.

Observe that it has been possible to conduct controlled simulations thanks to the efficiency and accuracy of SimRA [1]. Indeed, SimRA made the simulation of complex scenarios of multiple populations evolution (where each population has huge size) feasible.

From the promising results obtained, we believe that the topological signatures have the potential not only for detecting, but also discriminating ancient from recent admixture in multiple populations. This preliminary work is promising and in our future work, we plan to explore more complex admixture models, both in terms of complex topology of the scaffolds as well as complex characterizations of admixture.

## 8.3 Explaining evolution via Constrained Persistent Phylogeny - Ch. 6

The availability of a huge amount of genomic and proteomic data makes the use of genetic attributes or biological markers quite appealing in evolution analysis, thus giving even more importance to applying computationally efficient parsimony models. On the other hand, there is a huge gap between tractable and NP-hard parsimony models

that needs to be filled. In fact, one extreme is the perfect phylogeny, which has linear time solution but has only a few specific biological applications. On the other hand we have models, such as Dollo or Camin-Sokal parsimony models, which are often too generic from a biological viewpoint and computationally impracticable. A middle ground is occupied by the persistent perfect phylogeny model introduced in [66], which finds specific applications such as protein networks and domains analysis [61].

In [5, 6] we continue the investigation of the persistent perfect phylogeny model. More precisely, we investigate the constrained persistent perfect phylogeny (CPPP) problem, which is the general problem of computing a persistent perfect phylogeny for binary matrices where some characters may be forced to be non-persistent in the tree. We provide algorithmic solutions for the problem: a polynomial time algorithm when the conflict graph is edgeless and a fixed-parameter algorithm for the general CPPP problem (See Sections 6.3 and 6.4). Note that both the algorithms can be applied also to the unconstrained case, the PPP problem, simply posing the set of constrains $F = \emptyset$.

We experimentally illustrate that the search tree technique, combined with the use of constraints, speeds up the computation for matrices that otherwise would require exponential time. In [5] we run a preliminary experimental analysis showing that our method can manage successfully binary characters data incorporating back mutations. Indeed, the algorithm performs efficiently on simulated matrices as well as on real data taken from the HapMap project [69]. Hence, the experiments illustrate that the constrained persistent perfect phylogeny model allows to explain data that do not conform with the classical perfect phylogeny model (See Section 6.5). Future research can be devoted to experimental investigation of possible improvements based on introducing a carefully crafted set of constraints to speed up the computation.

Moreover, using the notion of red black graph, we give a polynomial time solution for both the constrained and unconstrained persistent perfect phylogeny reconstruction problems over instances that have edgeless conflict graphs [5]. Also we demonstrate that when no constraint is given and the conflict graph is edgeless, a solution for the PPP problem always exists.

Finally, we characterize an infinite class of instances on which the persistent phylogeny problem have no solution, but all their induced sub-instances can be solved by a persistent phylogeny [6]. These instances are represented by red black graphs with simple cycles of length greater than 4 and whose conflict graphs have also cycles (See Section 6.6). Clearly, graphs that contains such induced cyclic subgraphs are also not solvable. Hence, a natural question is the following. Is there a necessary and sufficient condition that characterizes the red black graphs that can be solved? Can such condition be tested in polynomial time?

Another relevant problem still open is whether there exists a polynomial time algorithm to find a successful reduction for a red black graph. Solving this question implies the possibility to develop a polynomial time solution for the PPP problem.

Observe, indeed, that the computational complexity of the CPPP problem, hence of the PPP problem, is still open. Moreover, as the CPPP problem is equivalent to the case of Generalized Cladistic Character Compatibility (GCCC) problem in Table 1 rows (5-6) in [64], whose complexity is left open, our results also apply to that case of GCCC problem.

It is also of interest considering the optimization version of the PPP problem, which consists of finding the persistent phylogeny that minimizes the number of persistent characters. Consequently, a natural question is characterizing the computational complexity of Parsimonious PPP problem.

Finally, from a theoretical point of view, the investigation of variants of the perfect phylogeny or restrictions of the Dollo parsimony, different from the persistent perfect phylogeny, is still an important research direction to be investigated.

## 8.4 Filling incomplete genomic sequences - Ch. 7

In [8] we consider two variants of the Scaffold Filling problem (One-sided SF-MNSA and the Two-sided SF-MNSA), a problem related to the reconstruction of complete genomes from incomplete draft genomes produced by the NGS technologies. We present two Fixed Parameter Tractable (FPT) algorithms for the two variants of Scaffold Filling, where the parameter is the number of common adjacencies in the resulting genomes (See Sections 7.3 and 7.4). However, the two designed FPT algorithms are only of theoretical relevance, since they have time complexity that makes them infeasible for practical applications. Therefore, there are some interesting open problems from an algorithmic perspective. First, it would be challenging to improve upon the time (and space) complexity of the two algorithms. In this direction, it would be interesting to investigate whether the algebraic technique applied to Graph Motif [89, 90] and Repetition Free Longest Common Subsequence [91] can be useful in this context. Then, it would be intriguing to study the kernelization complexity of the two problems. Moreover, the approximation complexity of the Scaffold Filling problems, in particular of the Two-sided case, should be further investigated. Finally, an interesting open problem in this direction (see [79]) is whether it is possible to design an approximation algorithm for Two-sided SF-MNSA with approximation factor better than 2.

# Bibliography


[1] A. P. Carrieri, F. Utro, and L. Parida. Sampling ARG of multiple populations under complex configurations of subdivision and admixture. *Bioinformatics*, 2015. doi: 10.1093/bioinformatics/btv716.

[2] A. P. Carrieri and L. Parida. SimRA: Rapid accurate simulation of populations based on random-graph models of ARG. RECOMB Comparative Genomics (RECOMB-CG) 2014, best poster award, Cold Spring Harbor (NY), 2014.

[3] L. Parida, F. Utro, D. Yorukoglu, A. P. Carrieri, D. Kuhn, and S. Basu. Topological signatures for population admixture. In T. M. Przytycka, editor, *Research in Computational Molecular Biology*, volume 9029 of *Lecture Notes in Computer Science*, pages 261–275. Springer International Publishing, 2015. doi: 10.1007/978-3-319-16706-0_27.

[4] P. Bonizzoni, A. P. Carrieri, G. Della Vedova, R. Dondi, and T. M. Przytycka. When and How the Perfect Phylogeny Model Explains Evolution. In N. Jonoska and M. Saito, editors, *Discrete and Topological Models in Molecular Biology, Natural Computing Series*, Natural Computing Series, pages 67–83. Springer-Verlag Berlin Heidelberg, Berlin, Germany, 2014. doi: 10.1007/978-3-642-40193-0__4.

[5] P. Bonizzoni, A. P. Carrieri, G. Della Vedova, and G. Trucco. Explaining evolution via constrained persistent perfect phylogeny. *BMC Genomics*, 15(6), 2014. doi: 10.1186/1471-2164-15-S6-S10.

[6] P. Bonizzoni, A. P. Carrieri, G. Della Vedova, and G. Trucco. A colored graph approach to the perfect phylogeny with persistent characters. Submitted, October 2015.

[7] L. Bulteau, A. P. Carrieri, and R. Dondi. Fixed-parameter algorithms for scaffold filling. In *Combinatorial Optimization - Third International Symposium, ISCO 2014, Lisbon, Portugal, March 5-7, 2014, Revised Selected Papers*, volume 8596, pages 137–148, 2014. doi: 10.1007/978-3-319-09174-7_12.

[8] L. Bulteau, A. P. Carrieri, and R. Dondi. Fixed-parameter algorithms for scaffold filling. *Theoretical Computer Science (TCS)*, 568:72–83, 2015. doi: 10.1016/j.tcs.2014.12.005.

[9] S. Hoban, G. Bertorelle, and O. E. Gaggiotti. Computer simulations: tools for population and evolutionary genetics. *Nature Reviews Genetics*, 13:110–122, 2012. doi: 10.1038/nrg3130.

[10] J. Hein, M. Schierup, and C. Wiuf. *Gene Genealogies, Variation and Evolution - A Primer in Coalescent Theory.* Wiley, 2004. ISBN 978-0-19-852996-5.

[11] M. B. Hamilton. *Population Genetics.* Wiley-Blackwell, 2009. ISBN 978-1-4051-3277-0.

[12] R. D. Hernandez. A flexible forward simulator for populations subject to selection and demography. *Bioinformatics*, 24(23):2786–2787, 2008. doi: 10.1093/bioinformatics/btn522.

[13] M. Chadeau-Hyam, C. J. Hoggart, P. F. O'Reilly, J. C. Whittaker, M. De Iorio, and D. J. Balding. Fregene: Simulation of realistic sequence-level data in populations and ascertained samples. *BMC Bioinformatics*, 9(364), 2008. doi: 10.1186/1471-2105-9-364.

[14] B. Peng and M. Kimmel. simuPOP: a forward-time population genetics simulation environment. *Bioinformatics*, 21(18):3686–3687, 2005. doi: 10.1093/bioinformatics/bti584.

[15] L. Parida. Nonredundant representation of ancestral recombinations graphs. In M. Anisimova, editor, *Evolutionary Genomics*, volume 856 of *Methods in Molecular Biology*, pages 315–332. Humana Press, 2012. doi: 10.1007/978-1-61779-585-5_13.

[16] S. Wright. Evolution in mendelian populations. *Genetics*, 16(2):97–159, 1931.

[17] R. A. Fisher. *The Genetical Theory of Natural Selection.* Clarendon Press, 1930.

[18] J. F. C. Kingman. On the genealogy of large populations. *Journal of Applied Probability*, 19:27–43, 1982. doi: 10.2307/3213548.

[19] L. Parida. Graph model of coalescence with recombinations. In L.S. Heath and N. Ramakrishnan, editors, *Problem Solving Handbook in Computational Biology and Bioinformatics*, pages 85–100. Springer US, 2010. doi: 10.1007/978-0-387-09760-2_5.

[20] R. R. Hudson. Properties of a neutral allele model with intragenic recombination. *Theoretical Population Biology*, 23(2):183–201, 1983. doi: doi:10.1016/0040-5809(83)90013-8.

[21] R. C. Griffiths and P. Marjoram. An ancestral recombinations graph. *Progress in Population Genetics and Human Evolution*, 87:257–270, 1997. doi: 10.1007/978-1-4757-2609-1_16.

[22] L. Parida. Ancestral Recombinations Graph: a reconstructability perspective using random-graphs framework. *Journal of Computational Biology*, 17(10):1345–1370, 2010. doi: 10.1089/cmb.2009.0243.

[23] S. Schaffner, C. Foo, S. Gabriel, D. Reich, M. Daly, and D. Altshuler. Calibrating a coalescent simulation of human genome sequence variation. *Genome Research*, 15: 1576–1583, 2005. doi: 10.1101/gr.3709305.

[24] C. C. Spencer and G. Coop. SelSim: a program to simulate population genetic data with natural selection and recombination. *Bioinformatics*, 12(20):3673–3675, 2005. doi: 10.1093/bioinformatics/bth417.

[25] G. Laval and L. Excoffier. SIMCOAL 2.0: a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history. *Bioinformatics*, 20(15):2485–2487, 2004. doi: 10.1093/bioinformatics/bth264.

[26] R. R. Hudson. Generating samples under a wright-fisher neutral model of 31 genetic variation. *Bioinformatics*, 18(2):337–338, 2002. doi: 10.1093/bioinformatics/18.2.337.

[27] L. Parida, P. Palamara, and A. Javed. A minimal descriptor of an ancestral recombinations graph. *BMC Bioinformatics*, 12(Suppl 1):S6, 2011. doi: 10.1186/1471-2105-12-S1-S6.

[28] F. Utro, M. Pybus, and L. Parida. Sum of parts is greater than the whole: inference of common genetic history of populations. *BMC Genomics*, 14(Suppl 1):S10, 2013. doi: 10.1186/1471-2164-14-S1-S10.

[29] S. R. S. Varadhan. *Probability theory*, volume 7 of *Courant Lecture Notes in Mathematics*. New York University, Courant Institute of Mathematical Sciences, New York; American Mathematical Society, Providence, RI, 2001. ISBN 0-8218-2852-5.

[30] M. Jobling, E. Hollox, M. Hurles, T. Kivisild, and C. Tyler-Smith. *Human evolutionary genetics*. Garland Science, UK, 2013.

[31] M.J. Kearsey and H.S. Pooni. *The Genetical Analysis of Quantitative Traits*. Stanley Thornes, 2004.

[32] M. Semon, R. Nielsen, M. P. Jones, and S. R. McCouch. The population structure of african cultivated rice oryza glaberrima (steud.) evidence for elevated levels of linkage disequilibrium caused by admixture with o. sativa and ecological adaptation. *Genetics*, 169(3):1639–1647, 2005. doi: 10.1534/genetics.104.033175.

[33] M. L. Freedman, C. A. Haiman, N. Patterson, G. .J. McDonald, A. Tandon, A. Waliszewska, K. Penney, R. G. Steen, K. Ardlie, E. M. John, I. Oakley-Girvan, A. S. Whittemore, K. A. Cooney, S. A. Ingles., D. Altshuler, B. E. Henderson, and D. Reich. Admixture mapping identifies 8q24 as a prostate cancer risk locus in african-american men. *Proceedings of the National Academy of Sciences*, 103(38): 14068–14073, 2006. doi: 10.1073/pnas.0605832103.

[34] J.D. Wall and M.F. Hammer. Archaic admixture in the human genome. *Current opinion in genetics & development*, 16(6):606–610, 2006. doi: 10.1016/j.gde.2006.09.006.

[35] P.-R. Loh, M. Lipson, N. Patterson, P. Moorjani, J. K. Pickrell, D. Reich, and B. Berger. Inferring admixture histories of human populations using linkage disequilibrium. *Genetics*, 193(4):1233–1254, 2013. doi: 10.1534/genetics.112.147330.

[36] B. D. Greenbaum, Li T.W. Olive, L. Poon, A. J. Levine, and R. Rabadan. Viral reassortment as an information exchange between viral segments. *Proceedings of the National Academy of Sciences*, 109(9):3341–3346, 2012. doi: 10.1073/pnas.1113300109.

[37] H. Edelsbrunner and J. L. Harer. *Computational topology.* American Mathematical Society, 2010. ISBN 978-0-8218-4925-5.

[38] E. H. Spanier. *Algebraic topology.* McGraw-Hill Book Co., 1966.

[39] H. Adams, A. Tausz, and M. Vejdemo-Johansson. javaplex: A research software package for persistent (co)homology. In H. Hong and C. Yap, editors, *Mathematical Software ICMS 2014*, volume 8592 of *Lecture Notes in Computer Science*, pages 129–136. Springer Berlin Heidelberg, 2014. doi: 10.1007/978-3-662-44199-2_23. URL `http://dx.doi.org/10.1007/978-3-662-44199-2_23`.

[40] S.R. Browning and B.L. Browning. Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. *The American Journal of Human Genetics*, 81(5):1084–1097, 2007. doi: 10.1086/521987.

[41] J. Felsenstein. *Inferring Phylogenies.* Sinauer Associates, 2004.

[42] L. L. Cavalli-Sforza and A. W. F. Edwards. Phylogenetic analysis: models and estimation procedures. *American journal of human genetics*, 19(3 Pt 1):233–257, 1967.

[43] T. M. Przytycka, G. Davis, N. Song, and D. Durand. Graph theoretical insights into evolution of multidomain proteins. *Journal of Computational Biology*, 13(2): 351–363, 2006. doi: 10.1089/cmb.2006.13.351.

[44] R. Schwartz and R. Shackney. Applying unmixing to gene expression data for tumor phylogeny inference. *BMC Bioinformatics*, 11(1):42, 2010. doi: 10.1186/1471-2105-11-42.

[45] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[46] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1): 19–28, 1991. doi: 10.1002/net.3230210104.

[47] I. Pe'er, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *Siam Journal on Computing*, 33(3):590–607, 2004. doi: 10.1137/S0097539702406510.

[48] P. Bonizzoni, G. Della Vedova, R. Dondi, and J. Li. The haplotyping problem: An overview of computational models and solutions. *Journal of Computer and Science Technology*, 18(6):675–688, 2003. doi: 10.1007/BF02945456.

[49] D. Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In *RECOMB*, pages 166–175, 2002. doi: 10.1145/565196.565218.

[50] P. Bonizzoni. A linear-time algorithm for the perfect phylogeny haplotype problem. *Algorithmica*, 48(3):267–285, 2007. doi: 10.1007/s00453-007-0094-3.

[51] Z. Ding, V. Filkov, and D. Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem. *Journal of Computational Biology*, 13(2):522–553, 2006. doi: 10.1089/cmb.2006.13.522.

[52] H. L. Bodlaender, M. R. Fellows, and T. J. Warnow. Two strikes against perfect phylogeny. In *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, pages 273–283, 1992. doi: 10.1007/3-540-55719-9_80.

[53] R. Agarwala and D. Fernández Baca. A polynomial time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM J. Comput.*, 23(6):1216–1224, 1994.

[54] S. Kannan and T. Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM J. Comput.*, 26(6):1749–1763, 1997. doi: 10.1137/S0097539794279067.

[55] A. Dress and M. Steel. Convex tree realizations of partitions. *Applied Mathematics Letters*, 5(3):3–6, 1992.

[56] S. Kannan and T. J. Warnow. Inferring evolutionary history from DNA sequences. *SIAM J. Comput.*, 23(4):713–737, 1994. doi: 10.1137/S0097539791222171.

[57] R. Gysel, F. Lam, and D. Gusfield. Constructing perfect phylogenies and proper triangulations for three-state characters. *Algorithms for Molecular Biology*, 7:26, 2012. doi: 10.1186/1748-7188-7-26.

[58] A. Subramanian, S. Shackney, and R. Schwartz. Inference of tumor phylogenies from genomic assays on heterogeneous samples. *Journal of Biomedicine and Biotechnology*, 2012:16, 2012. doi: 10.1155/2012/797812.

[59] K. E. V. Rens, V. Mäkinen, and A. I. Tomescu. SNV-PPILP: refined SNV calling for tumor data using perfect phylogenies and ILP. *Bioinformatics*, 31(7):1133–1135, 2015. doi: 10.1093/bioinformatics/btu755.

[60] J. Camil and R. Sokal. A method for deducting branching sequences in phylogenesy. *Evolution*, 19(3):311–326, 1965.

[61] T. M. Przytycka. An important connection between network motifs and parsimony models. In *Research in Computational Molecular Biology, 10th Annual International Conference, RECOMB 2006, Venice, Italy, April 2-5, 2006, Proceedings*, pages 321–335, 2006. doi: 10.1007/11732990_27.

[62] D. Fernández-Baca and J. Lagergren. A polynomial-time algorithm for near-perfect phylogeny. *SIAM J. Comput.*, 32(5):1115–1127, 2003. doi: 10.1137/S0097539799350839.

[63] R. V. Satya, A. Mukherjee, G. Alexe, L. Parida, and G. Bhanot. Constructing near-perfect phylogenies with multiple homoplasy events. *ISMB (Supplement of Bioinformatics)*, 22(14):514–522, 2006.

[64] J. Maňuch, M. Patterson, and A. Gupta. Towards a characterisation of the generalised cladistic character compatibility problem for non-branching character trees. In J. Chen, J. Wang, and A. Zelikovsky, editors, *Bioinformatics Research and Applications*, volume 6674 of *Lecture Notes in Computer Science*, pages 440–451. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-21260-4_41.

[65] J. Maňuch, M. Patterson, and A. Gupta. On the generalised character compatibility problem for non-branching character trees. In H. Ngo, editor, *Computing and Combinatorics*, volume 5609 of *Lecture Notes in Computer Science*, pages 268–276. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-02882-3_27.

[66] P. Bonizzoni, C. Braghin, R. Dondi, and G. Trucco. The binary perfect phylogeny with persistent characters. *Theoretical computer science*, 454:51–63, 2012. doi: 10.1016/j.tcs.2012.05.035.

[67] D. Gusfield, S. Eddhu, and C. Langley. Optimal efficient reconstruction of phylogenetic networks with constrained recombination. *Journal of Bioinformatics and Computational Biology*, 02(1):173–213, 2004.

[68] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Verlag, 1999.

[69] The International HapMap Consortium. A second generation human haplotype map of over 3.1 million SNPs. *Nature*, 449(7164):851–861, 2007. ISSN 0028-0836. doi: 10.1038/nature06258.

[70] X. Xia. *Comparative Genomics*. SpringerBriefs in Genetics, 2013.

[71] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of genome rearrangements*. The MIT Press, 2009.

[72] P. Chain, D. Grafham, R. Fulton, M. Fitzgerald, J. Hostetler, D. Muzny, J. Ali, et al. Genomics. Genome project standards in a new era of sequencing. *Science*, 326 (5950F):236–237, 2009. doi: 10.1126/science.1180614.

[73] A. Muñoz, C. Zheng, Q. Zhu, V. Albert, S. Rounsley, and D. Sankoff. Scaffold filling, contig fusion and gene order comparison. *BMC Bioinformatics*, 11:304, 2010. doi: 10.1186/1471-2105-11-304.

[74] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005. doi: 10.1093/bioinformatics/bti535.

[75] H. Jiang, C. Zheng, D. Sankoff, and B. Zhu. Scaffold filling under the breakpoint distance. In E. Tannier, editor, *Comparative Genomics*, volume 6398 of *Lecture Notes in Computer Science*, pages 83–92. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-16181-0_8.

[76] Z. Chen, B. Fu, R. Goebel, G. Lin, W. Tong, J. Xu, B. Yang, Z. Zhao, and B. Zhu. On the approximability of the exemplar adjacency number problem for genomes with gene repetitions. *Theoretical Computer Science*, 550:59–65, 2014. doi: 10.1016/j.tcs.2014.07.011.

[77] H. Jiang, C. Zheng, D. Sankoff, and B. Zhu. Scaffold filling under the breakpoint and related distances. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(4):1220–1229, 2012. doi: 10.1109/TCBB.2012.57.

[78] N. Liu, H. Jiang, D. Zhu, and B. Zhu. An improved approximation algorithm for scaffold filling to maximize the common adjacencies. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 10(4):905–913, 2013. doi: 10.1109/TCBB.2013.100.

[79] N. Liu and D. Zhu. The algorithm for the two-sided scaffold filling problem. In T-H.H. Chan, L. L. LapChi, and L. Trevisan, editors, *Theory and Applications of Models of Computation*, volume 7876 of *Lecture Notes in Computer Science*, pages 236–247. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38235-2. doi: 10.1007/978-3-642-38236-9_22.

[80] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[81] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. doi: 10.1145/210332.210337.

[82] M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Upper and lower bounds for finding connected motifs in vertex-colored graphs. *J. Comput. Syst. Sci.*, 77(4): 799–811, 2011. doi: 10.1016/j.jcss.2010.07.003.

[83] R. Dondi, G. Fertin, and S. Vialette. Complexity issues in vertex-colored graph pattern matching. *J. Discrete Algorithms*, 9(1):82–99, 2011. doi: 10.1016/j.jda.2010.09.002.

[84] N. Betzler, R. van Bevern, M. R. Fellows, C. Komusiewicz, and R. Niedermeier. Parameterized algorithmics for finding connected motifs in biological networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 8(5):1296–1308, 2011. doi: 10.1109/TCBB.2011.19.

[85] R. Dondi, G. Fertin, and S. Vialette. Finding approximate and constrained motifs in graphs. *Theor. Comput. Sci.*, 483:10–21, 2013. doi: 10.1016/j.tcs.2012.08.023.

[86] P. Bonizzoni, G. Della Vedova, R. Dondi, and Y. Pirola. Variants of constrained longest common subsequence. *Inf. Process. Lett.*, 110(20):877–881, 2010. doi: 10.1016/j.ipl.2010.07.015.

[87] P. Bonizzoni, R. Dondi, G. Mauri, and I. Zoppis. Restricted and swap common superstring: A multivariate algorithmic perspective. *Algorithmica*, 72(4):914–939, 2015. doi: 10.1007/s00453-014-9882-8.

[88] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi: 10.1137/0202019.

[89] A. Björklund, P. Kaski, and L. Kowalik. Probably optimal graph motifs. In *STACS*, pages 20–31, 2013. doi: 10.4230/LIPIcs.STACS.2013.20.

[90] S. Guillemot and F. Sikora. Finding and counting vertex-colored subtrees. *Algorithmica*, 65(4):828–844, 2013. doi: 10.1007/s00453-011-9600-8.

[91] G. Blin, P. Bonizzoni, R. Dondi, and F. Sikora. On the parameterized complexity of the repetition free longest common subsequence problem. *Inf. Process. Lett.*, 112(7): 272–276, 2012. doi: 10.1016/j.ipl.2011.12.009.