



JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

QUENCHED LIMITS OF COALESCENTS IN FIXED PEDIGREES

MASTER'S THESIS

submitted in partial fulfillment of the
requirements for the degree of Master of Science
to the Faculty of Physics, Mathematics and Computer Science
of the Johannes Gutenberg-University Mainz

on 2015-07-16 by

Andrey Tyukin

Supervisor: Prof. Dr. Matthias Birkner
Second reviewer: Prof. Dr. Achim Klenke

Abstract

A quenched limit theorem for coalescents in fixed pedigrees is proved. We investigate a Cannings model with Mendelian randomness. We consider panmictic populations with a fixed number of diploid individuals. We show that, under certain assumptions about the pair and triple coalescence probabilities, the laws of coalescents conditioned on the random pedigree converge stochastically to the law of the Kingman's n -coalescent. This result is additionally verified by computer simulations. Further experiments are conducted to investigate whether similar results might hold for more complex family models and populations of varying size.

German abstract

Wir untersuchen ein Cannings Modell mit Mendel'scher Vererbung, und betrachten panmiktische Populationen mit einer festen Anzahl von diploiden Individuen. Wir zeigen, dass Verteilungen von Koaleszenten, bedingt auf eine zufällige Umgebung, stochastisch gegen die Verteilung des Kingman-Koaleszenten konvergieren. Wir verifizieren unsere theoretischen Erkenntnisse mit Hilfe von Computersimulationen. Darüberhinaus untersuchen wir experimentell Modelle mit komplexeren Familienstrukturen und variierenden Populationsgrößen.

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Contents

1. Introduction	7
1.1. Background	7
1.2. Motivation	8
1.3. Organization of the thesis	10
2. Preliminaries	11
2.1. Sets and functions	11
2.2. Skorokhod space	12
2.3. Laplace Transform	13
3. Coalescents in Fixed Pedigrees	19
3.1. Cannings model with Mendelian randomness	19
3.2. Main result	20
3.3. States and holding times representation	23
3.4. State spaces	26
3.5. Functions Φ_a	32
3.6. Limiting behavior of two coalescents on common graph	41
3.7. Limiting behavior of a single coalescent	52
3.8. Convergence in Skorokhod space	54
3.9. Putting it all together	56
4. Simulations	61
4.1. Simulation framework	61
4.2. Complex family structures	62
4.2.1. Panmictic diploid model as monogamous haploid model	63
4.2.2. Monogamous families of diploid individuals	64
4.2.3. Polygynous fish	65
4.2.4. Eusocial insects	66
4.3. Varying population size	66
5. Conclusion	69
Appendices	71
A. Plots	73
B. Source code	79

1. Introduction

1.1. Background

Kingman's coalescent is a basic stochastic model that arises in population genetics. It can be used to model gene genealogies for a single locus, under the assumption that there is no mutation, no recombination, and that the genetic variation does not affect the fitness of individuals.

Kingman's coalescent can be obtained from the Wright-Fisher model [2]. For each natural number $g \in \mathbb{N}_0$, consider disjoint generations with N haploid individuals. Suppose that each individual from generation g chooses a parent from generation $g+1$ uniformly and independently. Now we can select n distinct individuals from the generation $g = 0$, assign an index $i \in \{1, \dots, n\}$ to each chosen individual, and track their ancestral lineages back into the past. Two lineages coalesce as soon as they hit the same individual. This process continues until all n lineages coalesce into a single lineage, that is, until the most recent common ancestor (MRCA) of the sample of n individuals is found. Figure 1.1 illustrates this construction.

The assignment of a predecessor from generation g to each index i from the set $\{1, \dots, n\}$ induces a partition of this set. Thus, we obtain a time-discrete partition-valued process. Accelerating the time by factor N yields a process that converges to a time-continuous partition-valued Markov chain, as N tends to infinity. This time-continuous Markov chain is the Kingman's n -coalescent. It simply starts with the finest possible partition, and then merges each pair of active lineages with rate 1. Figure 1.2 shows a realization of this process.

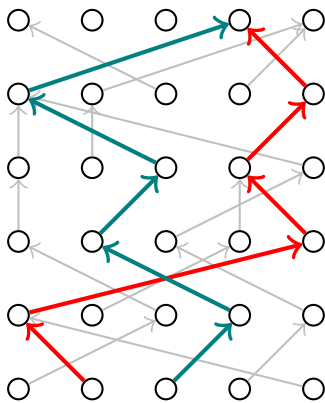


Figure 1.1.: Two coalescing lineages for sample size $n = 2$.

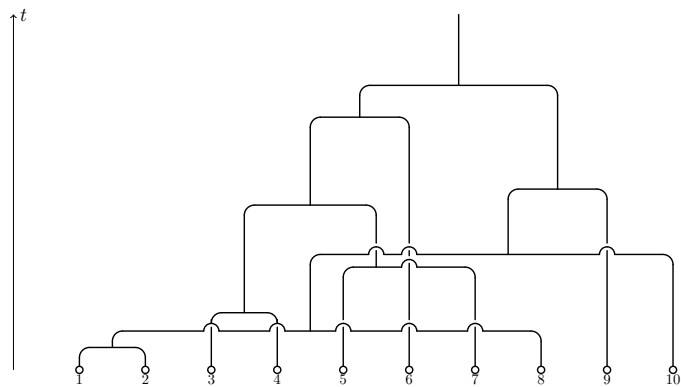


Figure 1.2.: A realization of Kingman's coalescent for sample size $n = 10$.

1. Introduction

1.2. Motivation

The model in the previous section is easy to formulate, and has a nice and elegant proof. However, real world applications where the Kingman's coalescent is used as a model can have seemingly vastly different assumptions. For example, Wakeley et al. considered gene genealogies in fixed pedigrees with diploid individuals [12]. All ancestral relationships were known and fixed. The only thing that was unknown (and thus modeled by a random variable), were the outcomes of the Mendelian inheritance experiments. It was known who the parents are, but it was not known which versions of chromosomes a child inherited from his/her parents. Thus, the assumptions in the application (fixed pedigree, Mendelian inheritance as the only source of randomness) are quite different from the assumptions used in the derivation of the Kingman's coalescent (random pedigree, no Mendelian randomness). Therefore, one could rightly doubt whether Kingman's coalescent is the most appropriate model in this particular case.

The following example shows that these doubts are not completely unfounded. Consider once again the standard Wright-Fisher model with haploid individuals and without any Mendelian randomness, where each individual chooses one parent from the previous generation uniformly. Suppose that we choose a population size N and a sample size n , generate a random pedigree, and fix it. Then, we generate multiple coalescents in this fixed pedigree, that is: we uniformly sample injections of the set $\{1, \dots, n\}$ into the set $\{1, \dots, N\}$, and then simply track the n selected lineages until we reach their MRCA. The result might look similar to what is shown in the Figure 1.3.

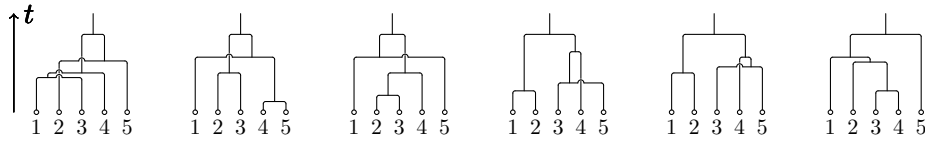


Figure 1.3.: Six coalescents in a fixed pedigree, no Mendelian randomness. Observe that the MRCA-times are the same in all these realizations.

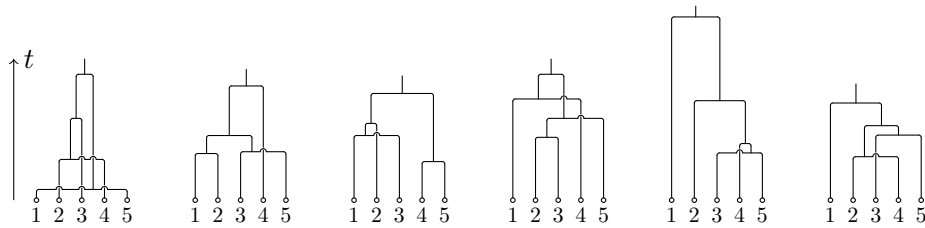


Figure 1.4.: Multiple realizations of the Kingman's coalescent. All coalescence times are distinct.

Each single coalescent in the Figure 1.3 looks like a typical realization of the

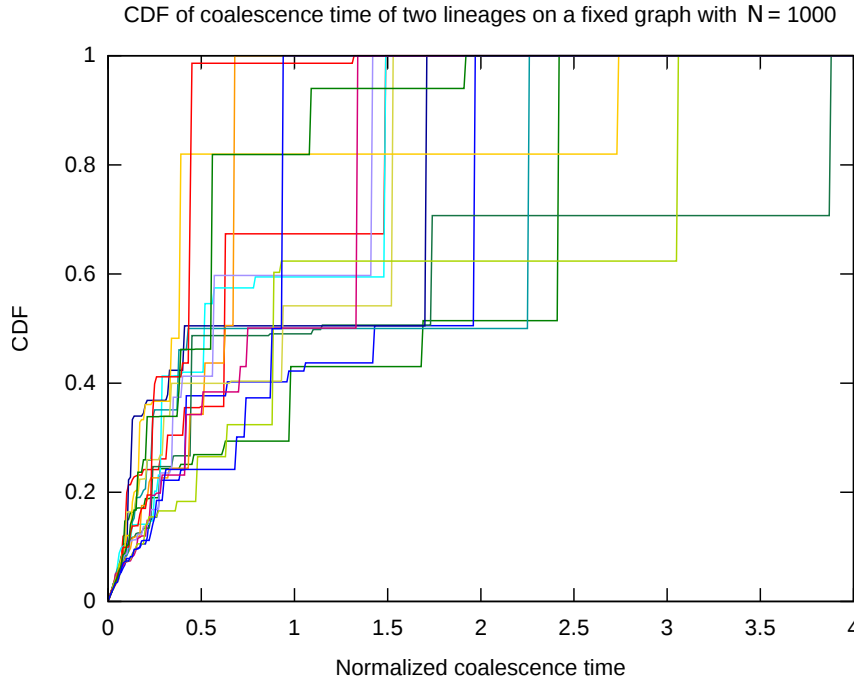


Figure 1.5.: Cumulative distribution function of the pair coalescence time on a fixed pedigree with haploid individuals and without any Mendelian randomness. The CDF is very different from the cumulative distribution function of Exp_1 .

Kingman's coalescent. However, if we consider all these coalescents *together*, we notice that their MRCA times coincide: this would be absolutely atypical for the Kingman's coalescent (compare Figure 1.3 to the Figure 1.4). It means that the distribution of these coalescents (conditioned on this particular fixed pedigree) is very different from the Kingman's coalescent. It is clear why it has to be different: the fixed pedigree is just a random tree, and all our coalescents simply end up at the (fixed!) MRCA of the entire population. The shape of the cumulative distribution function of the MRCA times for sample size $n = 2$ (Figure 1.5) confirms that the distribution of the coalescents on this fixed pedigree looks nothing like that of the Kingman's coalescent.

The question is: does Mendelian randomness change this situation fundamentally, or does it merely smooth and obfuscate the effect that is clearly visible in Figure 1.5? Is the Kingman's coalescent an adequate model if the pedigree is known and fixed? Fortunately, we can answer this question affirmatively: under fairly mild assumptions about the random process that generates the fixed pedigree, and with some Mendelian randomness, we can show that the distribution of coalescents in a fixed pedigree is likely to be not very different from the Kingman's coalescent. This partially explains experimental results obtained by Wakeley et al., which show that Kingman's coalescent provides a surprisingly accurate description of gene genealogies even if the underlying pedigree is fixed.

1.3. Organization of the thesis

The rest of this thesis is organized as follows. In Chapter 2 we introduce some notation, and briefly remind the reader of some properties of the Skorokhod space and the Laplace transform. In Chapter 3, we formulate the problem in terms of quenched limits of stochastic processes in random environments, and prove a quenched limit theorem for coalescents in fixed pedigrees. In Chapter 4, we describe a simulation framework and conduct several experiments, which indicate that the theorem from 3 might hold for more complex populations and family structures. Finally, we finish with a conclusion in Chapter 5.

2. Preliminaries

In this chapter, we fix the notation, and briefly remind the reader of some important definitions and theorems.

2.1. Sets and functions

We denote the *cardinality* of a set A by $\#A$. We denote the *powerset* of a set A by $\mathfrak{P}(A)$. Occasionally, we will specify a restriction on the cardinality of subsets in a subscript. In general, if P_1, \dots, P_n are some predicates on cardinal numbers, then we write

$$\mathfrak{P}_{P_1, \dots, P_n}(A) := \left\{ S \subset A : \bigvee_{i=1}^n P_i(\#S) \right\} \quad (2.1)$$

to denote the set of all subsets with cardinalities such that *at least one* of the predicates holds. We will often use intuitive abbreviations like “ ≥ 1 ” or “ $< \infty$ ”. For example, the expressions

$$\mathfrak{P}_{3,7}(A), \quad \mathfrak{P}_{\geq 1}(A), \quad \mathfrak{P}_{< \infty}(A), \quad \mathfrak{P}_{\leq \aleph_0}(A)$$

will denote, respectively, sets of subsets of A that

- have either exactly 3 or exactly 7 elements,
- are non-empty,
- are finite,
- are countable.

All functions from a set X to a set Y are denoted by Y^X . The cartesian product of sets $\{A_i\}_{i \in I}$ for some index set I is denoted by $\times_{i \in I} A_i$, the corresponding canonical projections from the cartesian product to A_i will be denoted by π_i , unless explicitly stated otherwise. We will often denote elements of cartesian products as $(a_i)_{i \in I}$ with $a_i \in A_i$, or just $(a_i)_i$ for short. If X is yet another set, and $f_i: X \rightarrow A_i$ are some functions, we denote the *product of functions* by

$$\begin{aligned} \langle f_i \rangle_{i \in I} : X &\rightarrow \times_{i \in I} A_i \\ x &\mapsto (f_i(x))_{i \in I}. \end{aligned} \quad (2.2)$$

2. Preliminaries

In particular, if $f: X \rightarrow A$ and $g: X \rightarrow B$, then $\langle f, g \rangle$ denotes just a componentwise defined function from X to $A \times B$. This should not be confused with some kind of scalar product (the standard scalar product on \mathbb{R}^n will be denoted as $\langle x, y \rangle_{\mathbb{R}^n}$).

The product of functions should also not be confused with the *cartesian product of functions*, defined as follows. Suppose that $f_i: A_i \rightarrow B_i$ for some sets A_i, B_i . Then we define:

$$\begin{aligned} \bigtimes_{i \in I} f_i: \bigtimes_{i \in I} A_i &\rightarrow \bigtimes_{i \in I} B_i \\ (a_i)_{i \in I} &\mapsto (f_i(a_i))_{i \in I}. \end{aligned} \quad (2.3)$$

We shall also occasionally use the notation $f^{\times k} := \bigtimes_{i=1}^k f$.

Some notational conventions are borrowed from combinatorics. Sets of all integers from 1 to n will be denoted as

$$[n] := \{1, \dots, n\}. \quad (2.4)$$

The set consisting of just the two elements $\{0, 1\}$ will be denoted as \mathbb{B} (for *Boolean*). For real n and natural k , the *falling factorial* is denoted by the *Pochhammer symbol*:

$$(n)_k := \prod_{i=0}^{k-1} (n - i) = n \cdot (n - 1) \cdot \dots \cdot (n - k + 1), \quad (2.5)$$

with the product having k terms in total.

2.2. Skorokhod space

Definition 2.2.1 (Skorokhod space). Let (E, ρ) be a metric space. Without loss of generality, assume that the distance between any two points $a, b \in E$ is not greater than 1, consider the truncated metric $\rho' := \rho \wedge 1$ instead of ρ if necessary.

We define the *Skorokhod space* $(D([0, \infty), E), d_{\text{Sk}})$ as follows. The carrier set $D([0, \infty), E)$ consists of all E -valued càdlàg functions, that is, functions x on $[0, \infty)$ with the following properties:

- For all $t > 0$ the left limit $x(t-) := \lim_{s \rightarrow t-} x(s)$ exists.
- For all $t \in [0, \infty)$ the right limit exists and is equal to the value of x at t :

$$x(t) = x(t+) := \lim_{s \rightarrow t+} x(s).$$

The metric d is defined as follows. Let Λ be the set of all strictly increasing homeomorphisms from $[0, \infty)$ onto $[0, \infty)$. For $\lambda \in \Lambda$ define

$$\gamma(\lambda) := \sup_{t > s \geq 0} \left| \log \left(\frac{\lambda(t) - \lambda(s)}{t - s} \right) \right|.$$

For $x, y \in D([0, \infty), E)$ set

$$d_{\text{Sk}}(x, y) := \inf_{\lambda \in \Lambda} \left(\gamma(\lambda) \vee \int_0^\infty e^{-u} d(x, y, \lambda, u) du \right), \quad (2.6)$$

where

$$d(x, y, \lambda, u) := \sup_{t \geq 0} \rho(x(t \wedge u), y(\lambda(t) \wedge u)).$$

Notice that for a $\lambda \in \Lambda$ it is possible that $\gamma(\lambda) = \infty$, however, such λ simply do not contribute anything to the infimum in the definition of d_{Sk} .

2.3. Laplace Transform

In this section we want to remind of some properties of the Laplace transform. All results from this section are standard, all main ideas can be found in a similar form for example in [4] and [1]. However, we use a Laplace transform on a space that is tailored to our specific problem.

We begin with a definition of a “customized” version of the Laplace transform for a space that looks like a disjoint union of multiple copies of $[0, \infty)^d$.

Definition 2.3.1. Let E be some finite set, and $d \in \mathbb{N}$ some dimension. For a finite measure $\mu \in \mathcal{M}_f(E \times [0, \infty)^d)$, we define the *Laplace transform* $\text{LT}_\mu : E \times [0, \infty)^d \rightarrow \mathbb{R}$ as follows:

$$\text{LT}_\mu(y, \lambda) := \int g_{y, \lambda} d\mu, \quad (2.7)$$

where the integrands $g_{y, \lambda}$ are real-valued functions on $E \times [0, \infty)^d$:

$$g_{y, \lambda}(x, t) := \mathbb{1}_{\{y\}}(x) e^{-\langle \lambda, t \rangle_{\mathbb{R}^d}}.$$

Remark 2.3.2. Since all $g_{y, \lambda}$ are continuous and bounded by 1, weak convergence of a sequence of measures on $E \times [0, \infty)^d$ implies pointwise convergence of the corresponding Laplace transforms. \diamond

The most of the rest of this section is devoted to the proof that the reverse implication also holds.

The first thing we want to verify is that the values of our Laplace transform uniquely determine a measure. The following lemma is a straightforward generalization of the one-dimensional case (see [4] Theorem 15.6).

Lemma 2.3.3. *The family of functions*

$$\begin{aligned} \mathcal{F} &:= \left\{ f_{S, \lambda} : S \subseteq E, \lambda \in [0, \infty)^d \right\}, \\ f_{S, \lambda} &: E \times [0, \infty)^d \rightarrow \mathbb{R}, \\ f_{S, \lambda}(x, t) &:= \mathbb{1}_S(x) e^{-\langle \lambda, t \rangle_{\mathbb{R}^d}} \end{aligned}$$

is separating for $\mathcal{M}_f(E \times [0, \infty)^d)$.

2. Preliminaries

Proof. Consider the one-point compactification $[0, \infty]$ of $[0, \infty)$, and let

$$c : [0, \infty) \rightarrow [0, \infty]$$

denote the Alexandroff extension. Define continuous functions

$$\begin{aligned} \tilde{f}_{S,\lambda} : E^d \times [0, \infty]^d &\rightarrow \mathbb{R} \\ (x, t) &\mapsto \mathbb{1}_S(x) \prod_{i=1}^d \psi(\lambda_i, t_i), \end{aligned}$$

where $\psi : [0, \infty) \times [0, \infty] \rightarrow \mathbb{R}$

$$\psi(\lambda, t) := \begin{cases} e^{-\lambda t} & \text{if } t < \infty \\ 1 & \text{if } t = \infty, \lambda = 0 \\ 0 & \text{if } t = \infty, \lambda > 0. \end{cases}$$

The family $\tilde{\mathcal{F}} = \{\tilde{f}_{S,\lambda}\}$ contains a constant non-zero function: $\tilde{f}_{E,0} = 1$. Because of

$$\psi(\lambda, t)\psi(\mu, t) = \psi(\lambda + \mu, t),$$

the family $\tilde{\mathcal{F}}$ is closed under pointwise multiplication:

$$\tilde{f}_{A,\lambda} \cdot \tilde{f}_{B,\mu} = \tilde{f}_{A \cap B, \lambda + \mu} \quad \forall A, B \subseteq E, \quad \lambda, \mu \in [0, \infty).$$

Finally, for any choice of two different elements $(x, t) \neq (y, s) \in E \times [0, \infty]^d$, there is a function $\tilde{f}_{S,\lambda}$ such that $\tilde{f}_{S,\lambda}(x, t) \neq \tilde{f}_{S,\lambda}(y, s)$: if $x \neq y$, we can simply take $\tilde{f}_{\{x\},0}$, otherwise $\tilde{f}_{E,\lambda}$ with any positive λ will do. Now, by a simple corollary of the Stone-Weierstrass theorem ([4] 15.3), it follows that $\tilde{\mathcal{F}}$ is separating for $\mathcal{M}_f(E \times [0, \infty]^d)$. Since the mapping

$$\begin{aligned} \mathcal{M}_f(E \times [0, \infty)^d) &\rightarrow \mathcal{M}_f(E \times [0, \infty]^d) \\ \mu &\mapsto \mu \circ c^{-1} \end{aligned}$$

is obviously injective, it follows from $f_{S,\lambda} = \tilde{f}_{S,\lambda} \circ c$ that \mathcal{F} is separating for $\mathcal{M}_f(E \times [0, \infty)^d)$. ■

Of course, we can compute the integral of $f_{S,\lambda}$ from the integrals of $g_{y,\lambda}$ for $y \in S$, therefore the Laplace transform contains all the information that is necessary to tell two different measures apart.

Corollary 2.3.4. *Every finite measure $\mu \in \mathcal{M}_f(E \times [0, \infty)^d)$ is uniquely determined by the values of LT_μ .*

2.3. Laplace Transform

Proof. Suppose $\text{LT}_\mu = \text{LT}_\nu$ for $\mu, \nu \in \mathcal{M}_f(E \times [0, \infty)^d)$. Then for each $f_{S,\lambda} \in \mathcal{F}$ it holds:

$$\int f_{S,\lambda} d\mu = \sum_{y \in S} \text{LT}_\mu(y, \lambda) = \sum_{y \in S} \text{LT}_\nu(y, \lambda) = \int f_{S,\lambda} d\nu. \quad (2.8)$$

By 2.3.3, $\mu = \nu$ must hold. ■

Now we are almost ready to prove that pointwise convergence of Laplace transforms implies weak convergence of measures. In the proof we will need the following well-known yet nameless statement from the elementary real analysis.

Lemma 2.3.5. *Fix some dimension $d \in \mathbb{N}$. For $x, y \in \mathbb{R}^d$, we write $x \leq y$ ($x < y$) if $x_i \leq y_i$ ($x_i < y_i$) for all $i \in [d]$. Let $q, p \in \mathbb{R}^d$ with $q < p$. Consider the compact rectangular box*

$$K := [q, p] := \left\{ x \in \mathbb{R}^d : q \leq x \leq p \right\}.$$

For each $n \in \mathbb{N} \cup \{\infty\}$, let $f_n : K \rightarrow \mathbb{R}$ be some functions that are non-increasing in the following sense:

$$x \leq y \quad \Rightarrow \quad f_n(x) \geq f_n(y)$$

for all $x, y \in K$. Suppose that $f := f_\infty$ is continuous and that $f_n \rightarrow f$ pointwise as $n \rightarrow \infty$. Then f_n converge to f uniformly.

Proof. Fix an arbitrarily small $\varepsilon > 0$. Since K is compact, f is uniformly continuous. Therefore, there exists a $\delta \geq 0$ such that for each $x \in K$ and each $y \in K$ the following implication holds:

$$\|x - y\|_\infty := \max_{i=1}^d |x_i - y_i| \leq \delta \quad \Rightarrow \quad |f(x) - f(y)| < \frac{\varepsilon}{2}. \quad (2.9)$$

The family of open (in the relative topology of K) cuboids

$$\mathcal{O} := \left\{ (a, b) \cap K : a, b \in \mathbb{R}^d, a < b, \|a - b\|_\infty < \delta \right\}$$

is an open covering of K , therefore there is an $N \in \mathbb{N}$ and cuboids C_i for $i \in [N]$ such that $\{C_i\}_{i=1}^N \subset \mathcal{O}$ is a finite covering of K . For each i , let a_i and b_i denote the vertices of the cuboid C_i , that is: $\bar{C}_i = [a_i, b_i]$, where \bar{C}_i denotes the closure of C_i . Since f_n converge pointwise to f , we can find an n_0 so large that the values of f_n at the selected vertices stay close enough to the values of f for all n beyond n_0 :

$$\max_{i=1}^N |f_n(a_i) - f(a_i)| < \frac{\varepsilon}{2}, \quad \max_{i=1}^N |f_n(b_i) - f(b_i)| < \frac{\varepsilon}{2}. \quad (2.10)$$

Now, for any $x \in K$, we can find an index $j \in [N]$ such that $x \in [a_j, b_j]$. The monotonicity property gives us upper and lower bounds for $f(x)$ and $f_n(x)$:

$$f_n(a_j) \geq f_n(x) \geq f_n(b_j),$$

2. Preliminaries

$$f(a_j) \geq f(x) \geq f(b_j).$$

If $f_n(x) \leq f(x)$, then from (2.9) and (2.10) we obtain:

$$f(x) - f_n(x) \leq f(a_j) - f_n(b_j) \leq |f(a_j) - f(b_j)| + |f(b_j) - f_n(b_j)| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

Swapping the roles of vertices a_j and b_j in the case $f_n(x) > f(x)$ gives an analogous estimation, so that $|f_n(x) - f(x)| < \varepsilon$ holds in all cases. Since the choice of $x \in K$ was arbitrary, we get

$$\|f_n - f\|_K := \sup_{x \in K} |f_n(x) - f(x)| < \varepsilon$$

for all $n \geq n_0$. Since the ε could be chosen arbitrarily small, this is exactly the definition of the uniform convergence. \blacksquare

Now we prove the central proposition of this section. We closely follow the proof strategy used by Billingsley ([1], Example 5.5).

Proposition 2.3.6. *Let $P_n, P \in \mathcal{M}_1(E \times [0, \infty)^d)$ such that the sequence $(LT_{P_n})_{n \in \mathbb{N}}$ converges pointwise to LT_P . Then the sequence $(P_n)_n$ is weakly convergent with*

$$\text{w-lim}_{n \rightarrow \infty} P_n = P.$$

Proof. Special case. First, consider a special case where E is a single element set: $E = \{*\}$.

We proceed in two steps. First, we show that the sequence of measures is tight. The second step is then a standard application of Prokhorov's theorem ([4] 13.29).

Part 1. Consider measures $(\mu_n)_n, \mu$ from $\mathcal{M}_{\leq 1}([0, \infty)^d)$. Identify $[0, \infty)^d$ with $\{*\} \times [0, \infty)^d$ and drop the first argument (constant $*$) in the notation of the Laplace transform for a moment. Suppose that the Laplace transforms LT_{μ_n} converge pointwise to LT_μ as n tends to infinity.

For $u \in \mathbb{R}_{>0}$ define

$$K_u := [0, u^{-1}]^d \subset [0, \infty)^d$$

and notice that this set is compact. The following little computation will be used towards the end of the subsequent chain of inequalities:

$$\int_{[0, u]^{-d}} \exp\left(-\sum_{i=1}^d t_i/u\right) dt = \left(\int_0^u e^{-\theta/u} d\theta\right)^d = \left([ue^{-\theta/u}]_0^u\right)^d = u^d(1 - e^{-1})^d. \quad (2.11)$$

Now, for any measure $\nu \in \mathcal{M}_f([0, \infty)^d)$, we can estimate how much mass is concentrated outside of K_u :

$$\frac{1}{u^d} \int_{[0, u]^{-d}} \nu\left([0, \infty)^d\right) - LT_\nu(t) dt = \frac{1}{u^d} \int_{[0, u]^{-d}} \int_{[0, \infty)^d} 1 - e^{-\langle x, t \rangle_{\mathbb{R}^d}} \nu[dx] dt$$

2.3. Laplace Transform

$$\begin{aligned}
&= \frac{1}{u^d} \int_{[0,\infty)^d} \int_{[0,u]^d} 1 - e^{-\langle x,t \rangle_{\mathbb{R}^d}} dt \nu[dx] \\
&\geq \frac{1}{u^d} \int_{K_u^c} \int_{[0,u]^d} 1 - \exp\left(-\sum_{i=1}^d t_i/u\right) dt \nu[dx] \\
&= \frac{1}{u^d} \int_{K_u^c} u^d - u^d(1 - e^{-1})^d \nu[dx] \\
&= \left(1 - (1 - e^{-1})^d\right) \nu[K_u^c].
\end{aligned}$$

Abbreviate $C_d := (1 - (1 - e^{-1})^d)$, and denote the integral on the left hand side by $I_u(\nu)$ for the rest of this proof.

Fix an arbitrarily small $\varepsilon > 0$. Since $\text{LT}_\mu(0_{d \times 1}) = \mu[[0, \infty)^d]$ and LT_μ is continuous, there must be an $u > 0$ so small that

$$\mu\left([0, \infty)^d\right) - \text{LT}_\mu(t) < \frac{\varepsilon C_d}{2}$$

for all $t \in [0, u]^d$, and thus $I_u(\mu) < \varepsilon C_d/2$. By lemma 2.3.5, LT_{μ_n} converge to LT_μ uniformly on the set $[0, u]^d$. Therefore, we can find $n_0 \in \mathbb{N}$ such that

$$\|\text{LT}_{\mu_n} - \text{LT}_\mu\|_{[0,u]^d} := \sup_{t \in [0,u]^d} |\text{LT}_{\mu_n}(t) - \text{LT}_\mu(t)| \leq \frac{\varepsilon C_d}{2}$$

and hence $|I_u(\mu_n) - I_u(\mu)| \leq \varepsilon C_d/2$ for all $n \geq n_0$. Putting all parts together, we obtain:

$$C_d \mu_n[K_u^c] \leq I_u(\mu_n) \leq I_u(\mu) + |I_u(\mu_n) - I_u(\mu)| \leq \frac{\varepsilon C_d}{2} + \frac{\varepsilon C_d}{2} = \varepsilon C_d,$$

that is $\mu_n[K_u^c] \leq \varepsilon$ for all $n \geq n_0$. Therefore, $\{\mu_n\}_{n \geq n_0}$ is tight. Since finite unions of tight families are again tight, the whole family $\{\mu_n\}_{n \in \mathbb{N}}$ is tight.

Part 2. Now we identify the weak limit of convergent subsequences of $\{\mu_n\}$.

Let $(\mu_{n_m})_m$ be an arbitrary subsequence of $(\mu_n)_n$. By Prokhorov's theorem we know that it contains a weakly convergent subsubsequence $(\mu_{n_{m_l}})_{l \in \mathbb{N}}$. Let ν be the weak limit of this subsubsequence. By remark 2.3.2, we know that $(\text{LT}_{\mu_{n_{m_l}}})_{l \in \mathbb{N}}$ converges pointwise to LT_ν . But the limit of $(\text{LT}_{\mu_{n_{m_l}}})_{l \in \mathbb{N}}$ is of course the same as the limit of $(\text{LT}_{\mu_n})_n$, namely LT_μ , that is $\text{LT}_\nu = \text{LT}_\mu$. By the lemma 2.3.4, it must hold $\nu = \mu$.

Now we know that every subsequence of $(\mu_n)_n$ contains a weakly convergent subsubsequence that converges to μ . By Urysohn's subsequence principle, $(\mu_n)_n$ itself also converges to μ .

General case. Now, instead of single-point set $\{*\}$ consider an arbitrary finite set E . For every fixed $y \in E$,

$$\mu_n^{(y)}(A) := P_n(\{y\} \times A), \quad \mu^{(y)}(A) := P(\{y\} \times A)$$

2. Preliminaries

are measures from $\mathcal{M}_{\leq 1}([0, \infty)^d)$ that fulfill the premises of the special case, therefore $\mu^{(y)} = \text{w-lim}_{n \rightarrow \infty} \mu_n^{(y)}$ must hold. For each $f \in C_b(E \times [0, \infty)^d)$, it holds:

$$\int f \, dP_n = \sum_{y \in E} \int f(y, t) \mu_n^{(y)}[dt] \xrightarrow{n \rightarrow \infty} \sum_{y \in E} \int f(y, t) \mu^{(y)}[dt] = \int f \, dP,$$

hence we obtain $P_n \Rightarrow P$. ■

We finish this section with a simple but useful corollary to the elementary lemma 2.3.5.

Corollary 2.3.7. *Let μ, μ^n be (sub)probability measures on $E \times [0, \infty)^d$ (with E, d as in 2.3.1). Suppose that $\mu[(E \times (0, \infty)^d)^c] = 0$ and that the Laplace transforms of μ_n converge to μ pointwise. Then the convergence of Laplace transforms is actually uniform.*

Proof. It is enough to consider the case where $E = \{*\}$ has just one element, the general case is a direct consequence. Fix an arbitrary $\varepsilon > 0$. For each $i \in [d]$, let $e_i := (\delta_{ij})_{j=1}^d$ denote the canonical basis vector of \mathbb{R}^d . Since $\mu[\{t_i = 0\}] = 0$, it holds by dominated convergence:

$$\lim_{u \rightarrow \infty} \text{LT}_\mu(ue_i) = \lim_{u \rightarrow \infty} \int_{(0, \infty)^d} e^{-ut_i} \mu[dt] = 0,$$

therefore we can find $u \in [0, \infty)$ so large that

$$\max_{i=1}^d \text{LT}_\mu(ue_i) < \frac{\varepsilon}{3}.$$

Since the Laplace transforms LT_{μ_n} are assumed to converge to LT_μ pointwise, we can find an $N \in \mathbb{N}$ so large that

$$\max_{i=1}^d |\text{LT}_{\mu_n}(ue_i) - \text{LT}_\mu(ue_i)| < \frac{\varepsilon}{3}$$

holds for all $n > N$. Fix a $\lambda \in ([0, u]^d)^c$. There is at least one index $i_0 \in [d]$ such that $\lambda \geq ue_{i_0}$. Since Laplace transforms are non-increasing, $\text{LT}_{\mu_n}(\lambda) \leq \text{LT}_{\mu_n}(ue_{i_0})$ holds (same with μ), and therefore:

$$|\text{LT}_{\mu_n}(\lambda) - \text{LT}_\mu(\lambda)| \leq |\text{LT}_{\mu_n}(ue_{i_0}) - \text{LT}_\mu(ue_{i_0})| + \text{LT}_\mu(ue_{i_0}) + |\text{LT}_\mu(\lambda)| < \frac{3\varepsilon}{3} = \varepsilon$$

for all $n > N$. Since this works for all $\lambda \in ([0, u]^d)^c$, we get uniform convergence on $([0, u]^d)^c$. From the lemma 2.3.5, we know that the convergence of LT_{μ_n} to LT_μ on $[0, u]^d$ is also uniform, therefore it is uniform on the entire space $[0, \infty)^d$. ■

3. Coalescents in Fixed Pedigrees

In this chapter, we formulate and prove a quenched limit theorem for coalescents in fixed pedigrees.

3.1. Cannings model with Mendelian randomness

We want to consider the simplest possible model where a coalescent on a fixed graph converges to the Kingman's coalescent. We assume that the population size is some constant $N \in \mathbb{N}$. There are disjoint generations with N diploid individuals in each generation $g \in \mathbb{N}_0$, where g should be thought of as the age of a generation. Each chromosome in the g -th generation is identified by an index of an individual $i \in \{1, \dots, N\} \equiv [N]$ and an index of the chromosome within the individual $c \in \{0, 1\} \equiv \mathbb{B}$. The number of chromosomes passed on to the generation of age $(g - 1)$ by the i -th individual from the generation g is determined by a \mathbb{N}_0 -valued random variable ν_{gi}^N . For each N and g , let $\nu_g^N = (\nu_{g,1}^N, \dots, \nu_{g,N}^N)$ be an independent copy of some random variable $\nu^N = (\nu_1^N, \dots, \nu_N^N)$ such that $\{\nu_1^N, \dots, \nu_N^N\}$ are exchangeable and sum up to $2N$. Furthermore, we assume that for each N and each generation g there is a uniformly chosen permutation σ_g^N of the set $[2N]$. This permutation models the fact that every diploid individual chooses two parents from the previous generation at random, thus our population is *panmictic*. The variables $(\nu_{gi}^N)_{gi}$ and $(\sigma_g^N)_g$ determine the structure of the random pedigree-graph, we therefore combine all these variables into a single variable \mathcal{G}^N :

$$\mathcal{G}^N := ((\nu_{gi}^N)_{gi}, (\sigma_g^N)_g)$$

We introduce the Mendelian randomness in the form of independent $\text{Ber}(1/2)$ -distributed binary values $m_g^N(i, c)$ for each generation $g \in \mathbb{N}_0$ and each chromosome $(i, c) \in [N] \times \mathbb{B}$. These values determine which one of the two chromosomes is inherited from the parent.

Now, suppose that there is a natural number $n \ll N$ (the sample size). Let \mathcal{I} for a moment denote the set of all injective functions from the set $\{1, \dots, n\} \equiv [n]$ into the set $[N]$. Using the random variables \mathcal{G}^N and m^N , we define a function-valued Markov chain $(X_g^{N,n})_g \equiv (X_g^{N,n}[\mathcal{G}^N, m^N])_g$, such that each realization of the random variable $X_g^{N,n}$ is a function from $[n]$ to $[N] \times \mathbb{B}$. Initially, we pick n different individuals, and let $X_0^{N,n}$ point to their chromosomes with index 0:

$$X_0^{N,n} := \langle U, \text{const}_0 \rangle \quad \text{with } U \sim \mathcal{U}_{\mathcal{I}}. \quad (3.1)$$

3. Coalescents in Fixed Pedigrees

Here, U is a uniformly chosen injection from $[n]$ to $[N]$, and const_0 is the constant 0 function from $[n]$ to \mathbb{B} . Thus, the morphism product (as introduced in section 2.1) is a random function from $[n]$ to $[N] \times \mathbb{B}$.

The transition from generation g to generation $g + 1$ is defined as follows:

$$X_{g+1}^{N,n} := \left\langle q[\nu_g^N] \circ \sigma_g^N \circ r, m_g^N \right\rangle \circ X_g^{N,n}. \quad (3.2)$$

Here r is a simple reshaping of the indices:

$$r(i, c) := c \cdot N + i \quad \text{for } i \in [N], c \in \mathbb{B},$$

and $q[\varphi]$ for $\varphi = (\varphi_1, \dots, \varphi_N)$ with $\varphi_k \in \mathbb{N}_0$ is a function from $[2N]$ to $[N]$ defined as follows:

$$q[\varphi](i) := \min \left\{ k \in [N] : \sum_{j=1}^k \varphi_j \geq i \right\}.$$

If each φ_i is interpreted as the number of chromosomes passed on to the next generation by i -th individual, then $q[\varphi](\sigma_g^N(r(i, c)))$ chooses an index of the parent for the (i, c) -th chromosome.

Throughout the entire chapter, we will use the following notation:

$$I(\varphi, j) := \left(\sum_{i=1}^{j-1} \varphi_i, \sum_{i=1}^j \varphi_i \right] \cap \mathbb{Z} \quad (3.3)$$

Notice that $I(\varphi, j)$ is a sequence of φ_j contiguous integers. Using this notation, we could have defined $q[\varphi](i)$ as the unique index j such that $i \in I(\varphi, j)$.

Notice that we suppress the underlying probability space in the notation: although $q[\nu_g^N]$, σ_g^N and $X_g^{N,n}$ are random variables, that is, measurable functions on some probability space, we always mean their *realizations* when we use function application and function composition. Realizations of the Mendelian random variables m_g^N are functions from $[N] \times \mathbb{B}$ to \mathbb{B} , realizations of $q[\nu_g^N] \circ \sigma_g^N \circ r$ are functions from $[N] \times \mathbb{B}$ to $[N]$, therefore their product (as defined in (2.2)) is an endomorphism of $[N] \times \mathbb{B}$, which composes just nicely with $([N] \times \mathbb{B})^{[n]}$ -valued realizations of $X_g^{N,n}$.

3.2. Main result

The goal of this section is to formulate the main result (Theorem 3.2.5). Before we can do this, we need a few more definitions.

Random variables $X_g^{N,n}$ are comparatively easy to define, but they contain too much irrelevant information. The following definition will allow us to forget some unnecessary details, and thereby bring $X_g^{N,n}$ into a common space even for different population sizes N .

Definition 3.2.1 (Partitions). Let A be an arbitrary set. By \mathcal{E}_A we denote the set of all possible *partitions* of A :

$$\mathcal{E}_A := \left\{ \{A_i\}_{i \in I} : I \text{ index set}, \emptyset \neq A_i \subseteq A \text{ pairwise disjoint}, \biguplus_{i \in I} A_i = A \right\}. \quad (3.4)$$

For natural n , we write $\mathcal{E}_n := \mathcal{E}_{[n]}$ for short. The finest possible partition of a set A is denoted by Δ_A :

$$\Delta_A := \{\{a\} : a \in A\} \in \mathcal{E}_A. \quad (3.5)$$

We will drop the subscript A if it can be inferred from the context.

If A, B are some sets, $f: A \rightarrow B$ some function, then we define the *partition induced by f* as follows:

$$\mathcal{E}(f) := \{f^{-1}(\{b\}) : b \in B\} \setminus \{\emptyset\} \in \mathcal{E}_A. \quad (3.6)$$

We will use the symbol \mathcal{E} for all such mappings from B^A to \mathcal{E}_A , regardless of what A and B are.

Finally, we equip the set \mathcal{E}_A with a relation \prec . For $\xi, \eta \in \mathcal{E}_A$, we write $\xi \prec \eta$ if there exists a partition $\tau \in \mathcal{E}_\xi$ such that

$$\eta = \left\{ \bigcup_{S \in F} S : F \in \tau \right\}.$$

Intuitively, this means that we can obtain η by merging some of the sets contained in ξ . Notice that this relation is a partial order.

We also write $\xi \vdash \eta$ if η arises from ξ by a pair-coalescence, more precisely: $\xi \vdash \eta$ if and only if $\xi = \{\xi_1, \dots, \xi_k\}$ for some $k \geq 2$ and $\eta = \{\xi_1 \cup \xi_2, \xi_3, \dots, \xi_k\}$, that is, $\xi \prec \eta$ and $\#\eta = \#\xi - 1$.

Definition 3.2.2. For every sample size $n \in \mathbb{N}$ and population size N we define \mathcal{E}_n -valued process $(\mathfrak{X}_g^{N,n})_g$ as follows:

$$\mathfrak{X}_g^{N,n} := \mathcal{E}(X_g^{N,n}).$$

All processes $\mathfrak{X}^{N,n}$ have values in the same space \mathcal{E}_n , but they seem to slow down as N gets larger. We account for this by rescaling the time parameter.

Definition 3.2.3 (Time scaling). For each $N \in \mathbb{N}$ denote the *pair* and *triple* coalescence probabilities by c_N and d_N respectively:

$$c_N := \frac{\mathbb{E}[(\nu_1^N)_2]}{4(2N-1)}, \quad d_N := \frac{\mathbb{E}[(\nu_1^N)_3]}{8(2N-1)(2N-2)}. \quad (3.7)$$

Now we can consider \mathcal{E}_n -valued time-continuous processes $(\mathfrak{X}_{[t/c_N]}^{N,n})_{t \in [0, \infty)}$. We will often write $\mathfrak{X}_{[-/c_N]}^{N,n}$ for short.

3. Coalescents in Fixed Pedigrees

Our ultimate goal will be to show that the process $\mathfrak{X}_{\lfloor t/c_N \rfloor}^{N,n}$, given the parentship graph \mathcal{G}^N , is very likely to have a distribution similar to the Kingman's coalescent for large population sizes N . A precise definition of the Kingman's coalescent is given below.

Definition 3.2.4 (Kingman's n -coalescent). Let $n \in \mathbb{N}$ be some sample size. *Kingman's n -coalescent* is a \mathcal{E}_n -valued time-continuous Markov chain with initial distribution

$$\mathbb{P}[\mathcal{K}_0^n = \Delta] = 1 \quad (3.8)$$

and Q-matrix

$$Q_{\xi\eta}^{(n)} := \begin{cases} 1 & \text{if } \xi \vdash \eta \\ -\binom{\#\xi}{2} & \text{if } \eta = \xi \\ 0 & \text{otherwise} \end{cases} \quad (3.9)$$

To formulate our main result, we need a suitable notion of convergence. For measures $\mu, (\mu^N)_N$ on the Skorokhod space $D([0, \infty), \mathcal{E}_n)$ we write

$$\mu^N \xrightarrow[N \rightarrow \infty]{\mathbb{P}} \mu \quad (3.10)$$

if μ^N converges in probability to μ with respect to the Lévy-Prokhorov metric d_{LP} :

$$\forall \varepsilon > 0 : \lim_{N \rightarrow \infty} \mathbb{P}[d_{LP}(\mu^N, \mu) > \varepsilon] = 0. \quad (3.11)$$

The Lévy-Prokhorov distance between two measures μ and ν on a metric space (M, d) is defined as follows:

$$d_{LP}(\mu, \nu) := \inf \{ \varepsilon > 0 : \mu(A) \leq \nu(A^\varepsilon) + \varepsilon, \nu(A) \leq \mu(A^\varepsilon) + \varepsilon \forall A \in \mathfrak{B}(\tau_d) \},$$

where $\mathfrak{B}(\tau_d)$ denotes the Borel σ -algebra generated by the topology induced by d and $A^\varepsilon = \{x \in M : \inf_{a \in A} d(x, a) < \varepsilon\}$ denotes the ε -fattening of the set A . This metric itself will not be particularly important here, what matters is that the weak convergence is equivalent to convergence with respect to d_{LP} if (M, d) is separable and complete (see [1] Theorem 6.8).

Now we can formulate the main result.

Theorem 3.2.5 (Quenched limit theorem for coalescents in fixed pedigrees). *Let \mathcal{G}^N as described in the modeling section 3.1 and $(\mathfrak{X}_{\lfloor t/c_N \rfloor}^{N,n})_t$ as defined in 3.2.3. Suppose that c_N as well as d_N/c_N converge to 0 as $N \rightarrow \infty$. Then it holds:*

$$\mathcal{L} \left(\left(\mathfrak{X}_{\lfloor t/c_N \rfloor}^{N,n} \right)_t \middle| \mathcal{G}^N \right) \xrightarrow[N \rightarrow \infty]{\mathbb{P}} \mathcal{L}(\mathcal{K}^n). \quad (3.12)$$

The rest of the entire chapter will be devoted to the proof of this theorem.

3.3. States and holding times representation

Neither the Lévy-Prokhorov metric, nor the Skorokhod metric are particularly convenient to work with directly. We therefore translate statements concerning those metrics into more straightforward statements about convergence of simple discrete and real-valued random variables.

The very first thing we want to do is to express weak convergence of processes in $D([0, \infty), \mathcal{E}_n)$ in terms of weak convergence of states and holding times in a much simpler space $\mathcal{E}_n^{n-1} \times [0, \infty)^{n-1}$.

Definition 3.3.1 (States and holding times representation). Let $D^\downarrow([0, \infty), \mathcal{E}_n)$ denote the subspace of those càdlàg functions $(y_t)_t$ for which $(\#y_t)_t$ is nonincreasing and y_t converges to the trivial partition $\{[n]\}$ as $t \rightarrow \infty$. For each $n \in \mathbb{N}$ we define functions

$$\Theta : D^\downarrow([0, \infty), \mathcal{E}_n) \rightarrow \mathcal{E}_n^{n-1} \times [0, \infty)^{n-1} \quad (3.13)$$

and their inverses Θ^{-1} by the following construction. Let $(y_t)_t$ be some function from $D^\downarrow([0, \infty), \mathcal{E}_n)$. For each $k \in [n]$ define times T_k as follows:

$$T_n := 0 \quad (3.14)$$

$$T_k := \inf \{t \geq T_{k+1} : \#y_t \leq k\} \quad (3.15)$$

Denote the differences between T_k by H_k , that is, for each $k \in \{2, \dots, n\}$ set:

$$H_k := T_{k-1} - T_k. \quad (3.16)$$

Furthermore, define $S_k \in \mathcal{E}_n$ for each $k \in [n]$ by

$$S_k := y_{T_k}. \quad (3.17)$$

The mapping Θ can now be defined as the assignment of states S_k and holding times H_k to the function y :

$$\Theta(y) := ((S_k)_{k=2}^n, (H_k)_{k=2}^n) \in \mathcal{E}_n^{n-1} \times [0, \infty)^{n-1}. \quad (3.18)$$

The inverse mapping Θ^{-1} is defined as follows. Given states $(S_k)_{k=2}^n$ and holding times $(H_k)_{k=2}^n$ we can of course easily reconstruct times T_k for each $k \in [n]$:

$$T_k := \sum_{i=k+1}^n H_i, \quad (3.19)$$

which in turn allows us to recover the entire function y :

$$y_t := S_{\min\{k \in [n] : T_k \leq t\}}. \quad (3.20)$$

3. Coalescents in Fixed Pedigrees

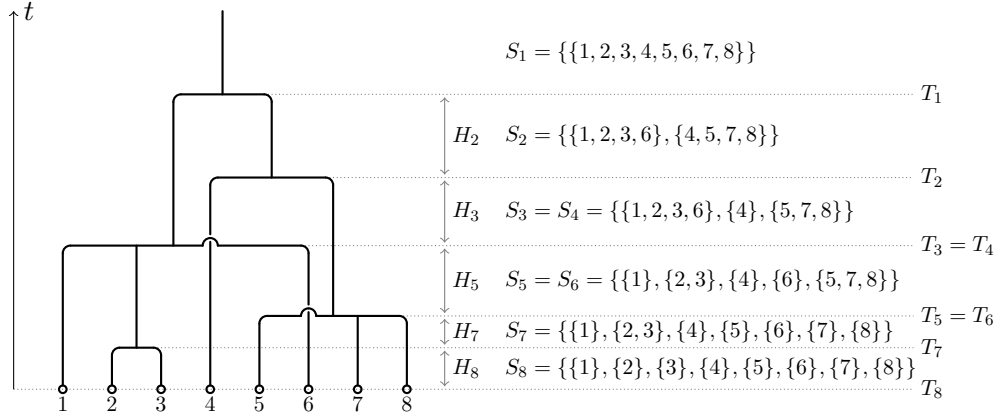


Figure 3.1.: A possible realization of the process $(\mathfrak{X}_{[t/c_N]}^{N,n})_t$ for sample size $n = 8$. The states S_k and times T_k from the definition 3.3.1 are shown on the right. Some of the H_k are also shown (the hidden ones are 0).

The holding times H_k tell us how much time the function y spends in the state S_k with k active lineages before jumping to the state S_{k-1} . This definition might look somewhat counterintuitive on first glance, because the times and states are indexed by the *decreasing* number of elements in the partition (each such element corresponding to an active lineage of the coalescent), so that we are counting backwards. The following example illustrates the definitions of Θ and Θ^{-1} .

Example 3.3.2 (States and holding times). Consider the realization of the process $(\mathfrak{X}_{[t/c_N]}^{N,n})_t$ shown in the Figure 3.1. The values

$$(S, H) = ((S_k)_{k=2}^n, (H_k)_{k=2}^n) = \Theta \left(\left(\mathfrak{X}_{[t/c_N]}^{N,n} \right)_t \right)$$

(except those H_k that are equal 0) are shown on the right. We want to demonstrate the evaluation of Θ^{-1} on a few simple cases.

- Suppose that $t \in (T_1, T_2)$. We want to compute $\Theta^{-1}(S, H)(t)$. Clearly,

$$\min \{k : T_k \leq t\} = 2,$$

therefore $\Theta^{-1}(S, H)(t) = S_2$.

- Now suppose that $t \in (T_3, T_6)$. This time,

$$\min \{k : T_k \leq t\} = 5,$$

therefore $\Theta^{-1}(S, H)(t) = S_5$.

- Finally, let's see what happens in points of discontinuity. For example, consider $t = T_3 = T_4$. It holds:

$$\min \{k : T_k \leq t\} = 3,$$

3.3. States and holding times representation

and we obtain $\Theta^{-1}(S, H)(T_3) = S_3$, so that $\Theta^{-1}(S, H)$ is càdlàg at T_3 .

We conclude that at least for this special case our definition of Θ^{-1} behaves as expected. \diamond

Remark 3.3.3. Notice that, in contrast to $D([0, \infty), \mathcal{E}_n)$, the subspace $D^\perp([0, \infty), \mathcal{E}_n)$ is separable (we can obtain a countable dense subset by enumerating all combinations of $n - 1$ partitions, and forcing the holding times to be rational). Since it is closed, it is also complete. Thus, convergence in the Lévy-Prokhorov metric restricted to $D^\perp([0, \infty), \mathcal{E}_n)$ is equivalent to weak convergence. \diamond

As we will see later in section 3.8, for our special case the weak convergence in $D^\perp([0, \infty), \mathcal{E}_n)$ is equivalent to weak convergence of the corresponding $\mathcal{E}_n^{n-1} \times [0, \infty)^{n-1}$ -valued random variables. The proposition 2.3.6 in turn suggests that it is enough to control the Laplace transforms in order to ensure weak convergence. We will achieve this by showing that the expected values of the (random, graph-dependent) Laplace transforms converge to a known Laplace transform closely related to the Kingman's coalescent, and that the variance tends to zero. In order to control the variance, we need the following device.

Lemma 3.3.4. *Let E be a finite set, $d \in \mathbb{N}$ a dimension, and Y, \hat{Y}, \check{Y} random variables on some probability space $(\Omega, \mathcal{A}, \mathbb{P})$ with values in $E \times [0, \infty)^d$. Denote components ("states" and "times") of \hat{Y} and \check{Y} by (\hat{S}, \hat{H}) and (\check{S}, \check{H}) respectively. Let $\mathcal{F} \subset \mathcal{A}$ be a σ -algebra. Suppose that $\mathcal{L}(Y|\mathcal{F}) = \mathcal{L}(\hat{Y}|\mathcal{F}) = \mathcal{L}(\check{Y}|\mathcal{F})$. Moreover, suppose that \hat{Y} and \check{Y} are conditionally independent given \mathcal{F} . Then it holds for all $y \in E$ and $\lambda \in [0, \infty)^d$:*

1. $\mathbb{E}[\text{LT}_{\mathcal{L}(Y|\mathcal{F})}(y, \lambda)] = \text{LT}_{\mathcal{L}(Y)}(y, \lambda)$
2. $\text{Var}[\text{LT}_{\mathcal{L}(Y|\mathcal{F})}(y, \lambda)] = \text{LT}_{\mathcal{L}((\hat{S}, \check{S}), \hat{H} + \check{H})}((y, y), \lambda) - (\text{LT}_{\mathcal{L}(Y)}(y, \lambda))^2$

Proof. The first equation follows immediately from the definition 2.3.1 and the tower-property of the conditional expectation:

$$\begin{aligned} \mathbb{E}[\text{LT}_{\mathcal{L}(Y|\mathcal{F})}(y, \lambda)] &= \mathbb{E}[\mathbb{E}[g_{y, \lambda}(Y) | \mathcal{F}]] \\ &= \mathbb{E}[g_{y, \lambda}(Y)] \\ &= \text{LT}_{\mathcal{L}(Y)}(y, \lambda). \end{aligned}$$

The second equation follows from the tower-property together with conditional independence and the definition of $g_{y, \lambda}$ from 2.3.1. It holds:

$$\begin{aligned} \mathbb{E}[(\text{LT}_{\mathcal{L}(Y|\mathcal{F})}(y, \lambda))^2] &= \mathbb{E}[\text{LT}_{\mathcal{L}(\hat{Y}|\mathcal{F})}(y, \lambda) \text{LT}_{\mathcal{L}(\check{Y}|\mathcal{F})}(y, \lambda)] \quad (3.21) \\ &= \mathbb{E}[\mathbb{E}[g_{y, \lambda}(\hat{Y}) | \mathcal{F}] \mathbb{E}[g_{y, \lambda}(\check{Y}) | \mathcal{F}]] \\ &= \mathbb{E}[\mathbb{E}[g_{y, \lambda}(\hat{Y}) \cdot g_{y, \lambda}(\check{Y}) | \mathcal{F}]] \end{aligned}$$

3. Coalescents in Fixed Pedigrees

$$\begin{aligned}
&= \mathbb{E} \left[\mathbb{1}_{\{y\}}(\hat{S}) \mathbb{1}_{\{y\}}(\check{S}) e^{-\langle \lambda, \hat{H} + \check{H} \rangle_{\mathbb{R}^d}} \right] \\
&= \mathbb{E} \left[g_{(y,y),\lambda} \left((\hat{S}, \check{S}), \hat{H} + \check{H} \right) \right] \\
&= \text{LT}_{\mathcal{L}((\hat{S}, \check{S}), \hat{H} + \check{H})}((y, y), \lambda).
\end{aligned}$$

This, together with the first statement, entails the second part. \blacksquare

The above lemma suggests to consider two copies $\hat{X}^{N,n}$, $\check{X}^{N,n}$ of the process $X^{N,n}$ on the same random graph \mathcal{G}^N .

Definition 3.3.5 (Twin processes on common graph). Let N, n, \mathcal{G}^N as above. For each N , consider two independent families of $\text{Ber}(1/2)$ -distributed random variables \hat{m}_g^N and \check{m}_g^N (defined analogously to m_g^N in the section 3.1). Define two processes on common graph

$$(\hat{X}_g^{N,n})_g \equiv (\hat{X}_g^{N,n}[\mathcal{G}^N, \hat{m}^N])_g, \quad (\check{X}_g^{N,n})_g \equiv (\check{X}_g^{N,n}[\mathcal{G}^N, \check{m}^N])_g \quad (3.22)$$

analogously to $(X_g^{N,n})_g$. Furthermore, let $\hat{\mathfrak{X}}_g^{N,n}$, $\check{\mathfrak{X}}_g^{N,n}$ be analogous to the definition 3.2.2, and $\hat{\mathfrak{X}}_{[t/c_N]}^{N,n}$, $\check{\mathfrak{X}}_{[t/c_N]}^{N,n}$ analogous to the construction in 3.2.3.

The overall strategy is to apply lemma 3.3.4 to $E := \mathcal{E}_n^{n-1}$,

$$Y = \Theta((\hat{\mathfrak{X}}_{[t/c_N]}^{N,n})_t), \quad \hat{Y} = \Theta((\hat{\mathfrak{X}}_{[t/c_N]}^{N,n})_t), \quad \check{Y} = \Theta((\check{\mathfrak{X}}_{[t/c_N]}^{N,n})_t)$$

and $\mathcal{F} := \sigma(\mathcal{G}^N)$. This will allow us to control the variance of the Laplace transform of $\mathcal{L}(\Theta(\hat{\mathfrak{X}}_{[-/c_N]}^{N,n})|\mathcal{G}^N)$. Controlling the second moment (3.21) is the difficult part, it is the topic of the next three sections.

By the time we can control the second moment, we will have enough tools to calculate the Laplace transform of $\Theta(\hat{\mathfrak{X}}_{[-/c_N]}^{N,n})$. This is the much easier part, we will defer it until section 3.7.

3.4. State spaces

The processes $\hat{X}^{N,n}$ and $\check{X}^{N,n}$ contain all the available information about the random lineages, but they are unsuitable for discussing convergence, because they take values in different spaces for different population sizes N . We need a suitable common state space that does not depend on the population size, but still captures the dependence between the both components of the process $(\hat{X}_g^{N,n}, \check{X}_g^{N,n})_g$.

Definition 3.4.1 (State space for coalescents on same graph). For a given sample size $n \in \mathbb{N}$ define the state space \mathcal{H}_n as follows:

$$\mathcal{H}_n := \left\{ \xi \subset \mathfrak{P}_{1,2} \left(\mathfrak{P}([n])^2 \setminus \left\{ \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix} \right\} \right) : \biguplus_{I \in \xi} \biguplus_{c \in I} c_k = [n] \text{ for } k = 1, 2 \right\}. \quad (3.23)$$

3.4. State spaces

Each set $I \in \xi$ stands for an individual. Each element c of an individual stands for a chromosome. The index k distinguishes between the first coalescent and the second coalescent.

If $\xi \in \mathcal{H}_n$ is an element of our state space, $I \in \xi$ is an individual, and $c \in I$ is a chromosome, then we will use the suggestive notation

$$\hat{c} := \pi_1(c) \equiv c_1, \quad \check{c} := \pi_2(c) \equiv c_2 \in \mathfrak{P}([n])$$

to denote the components of c . We will also identify the indices $\{1, 2\}$ with symbols $\{\wedge, \vee\}$ and use a dot instead of an additional subscript, so that we can for example write $c = (\dot{c})_{\bullet \in \{\wedge, \vee\}}$. This shall emphasize the connection between the components of the chromosomes and the corresponding coalescents $\hat{\mathfrak{X}}^{N,n}$ and $\check{\mathfrak{X}}^{N,n}$.

For $\xi \in \mathcal{H}_n$, define:

$$\xi' := \bigcup_{I \in \xi} I. \quad (3.24)$$

This is the set of all chromosomes in ξ , without the boundaries between individuals.

Elements of \mathcal{H}_n should be thought of as little data structures that can hold information about two partitions of $[n]$ simultaneously, and also represent all the short-lived interferences and entanglements that occasionally occur between the two processes $\hat{X}^{N,n}$ and $\check{X}^{N,n}$.

Now we need some mapping that extracts all the relevant information from the realizations of $(\hat{X}_g^{N,n}, \check{X}_g^{N,n})$ and yields an element of the space \mathcal{H}_n .

Definition 3.4.2. For two functions $f, g : [n] \rightarrow [N] \times \mathbb{B}$, define:

$$\mathcal{H}(f, g) := \left\{ \left\{ \begin{bmatrix} f^{-1}(i, c) \\ g^{-1}(i, c) \end{bmatrix} : c \in \mathbb{B} \right\} \setminus \left\{ \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix} \right\} : i \in [N] \right\} \setminus \{\emptyset\}. \quad (3.25)$$

This is a function from $(([N] \times \mathbb{B})^{[n]})^2$ to \mathcal{H}_n , which we denote with the same symbol \mathcal{H} , but without any indices.

Now we can define a \mathcal{H}_n -valued process $\mathfrak{Z}^{N,n}$ that captures the dependence of two coalescents.

Definition 3.4.3. For each nonnegative integer g define:

$$Z_g^{N,n} := \left(\hat{X}_g^{N,n}, \check{X}_g^{N,n} \right), \quad \mathfrak{Z}_g^{N,n} := \mathcal{H}(Z_g^{N,n}),$$

and write $\mathfrak{Z}^{N,n} := (\mathfrak{Z}_g^{N,n})_{g \in \mathbb{N}_0}$ for short.

Ultimately, we will want to make statements about convergence of $\mathcal{E}_n \times \mathcal{E}_n$ -valued processes, therefore we have to establish a connection between \mathcal{H}_n and $\mathcal{E}_n \times \mathcal{E}_n$.

3. Coalescents in Fixed Pedigrees

Definition 3.4.4. For a given sample size $n \in \mathbb{N}$, define two functions

$$\rho : \mathcal{H}_n \rightarrow \mathcal{E}_n \times \mathcal{E}_n, \quad \iota : \mathcal{E}_n \times \mathcal{E}_n \rightarrow \mathcal{H}_n$$

as follows:

$$\rho(\chi) := (\pi_1(\chi') \setminus \{\emptyset\}, \pi_2(\chi') \setminus \{\emptyset\}), \quad (3.26)$$

$$\iota(\xi, \eta) := \left\{ \left\{ \begin{bmatrix} x \\ \emptyset \end{bmatrix} \right\} : x \in \xi \right\} \cup \left\{ \left\{ \begin{bmatrix} \emptyset \\ y \end{bmatrix} \right\} : y \in \eta \right\}, \quad (3.27)$$

where π_1 and π_2 are the canonical projections from $\mathfrak{P}([n])^2$ to $\mathfrak{P}([n])$. Sometimes, we shall also write

$$\begin{aligned} \rho_1(\chi) &:= \pi_1(\chi') \setminus \{\emptyset\} \\ \rho_2(\chi) &:= \pi_2(\chi') \setminus \{\emptyset\} \end{aligned} \quad (3.28)$$

to denote components of ρ separately.

Intuitively, the function ρ forgets the boundaries between individuals and cuts all chromosomes asunder, sorting active lineages that belong to the first and second coalescent into the first and the second partition respectively.

The function ι embeds the product $\mathcal{E}_n \times \mathcal{E}_n$ into \mathcal{H}_n by putting each active lineage into a separate chromosome of a separate individual.

The following example illustrates the definitions just introduced.

Example 3.4.5. Suppose that the population size is $N = 100$, the sample size is $n = 5$, and that (\hat{x}, \tilde{x}) is a realization of $(\hat{X}_g^{N,n}, \tilde{X}_g^{N,n})$ with values specified by the following table:

k	1	2	3	4	5
$\hat{x}(k)$	(23, 0)	(23, 0)	(59, 0)	(59, 1)	(17, 1)
$\tilde{x}(k)$	(17, 1)	(17, 0)	(59, 0)	(17, 1)	(59, 0)

Forgetting unnecessary details like individual and chromosome indices yields the following element $\xi := \mathcal{H}(\hat{x}, \tilde{x})$ of \mathcal{H}_n :

$$\xi = \left\{ \left\{ \begin{bmatrix} \{1, 2\} \\ \emptyset \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \{5\} \\ \{1, 4\} \end{bmatrix}, \begin{bmatrix} \emptyset \\ \{2\} \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \{4\} \\ \emptyset \end{bmatrix}, \begin{bmatrix} \{3\} \\ \{3, 5\} \end{bmatrix} \right\} \right\}.$$

Removing boundaries between individuals, and splitting each chromosome into two components (one for \hat{x} , one for \tilde{x}), yields:

$$\rho(\xi) = (\{\{1, 2\}, \{3\}, \{4\}, \{5\}\}, \{\{1, 4\}, \{2\}, \{3, 5\}\}).$$

Applying the function ι does not restore the boundaries between the individuals. Seven different individuals are generated instead, one for each set in the both partitions:

$$\iota(\rho(\xi)) = \left\{ \left\{ \begin{bmatrix} \{1, 2\} \\ \emptyset \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \{3\} \\ \emptyset \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \{4\} \\ \emptyset \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \{5\} \\ \emptyset \end{bmatrix} \right\} \right\} \cup$$

$$\left\{ \left\{ \left[\begin{array}{c} \emptyset \\ \{1, 4\} \end{array} \right] \right\}, \left\{ \left[\begin{array}{c} \emptyset \\ \{2\} \end{array} \right] \right\}, \left\{ \left[\begin{array}{c} \emptyset \\ \{3, 5\} \end{array} \right] \right\} \right\}$$

◇

Remark 3.4.6. The function ι is injective, ρ is surjective, and it holds:

$$\rho \circ \iota = \text{Id}_{\mathcal{E}_n \times \mathcal{E}_n}.$$

All these functions relate to \mathcal{E} introduced in 3.2.1 as follows:

$$\rho \circ \mathcal{H} = \mathcal{E} \times \mathcal{E},$$

where $\mathcal{E} \times \mathcal{E}$ denotes the cartesian product of functions.

The following diagram summarizes the relationships between the various state spaces:

$$\begin{array}{ccc} \left(([N] \times \mathbb{B})^{[n]} \right)^2 & \xrightarrow{\mathcal{H}} & \mathcal{H}_n \\ & \searrow \mathcal{E} + \mathcal{E} & \downarrow \rho \\ & & \mathcal{E}_n \times \mathcal{E}_n \end{array} \quad \begin{array}{c} \uparrow \iota \end{array}$$

Notice that this diagram commutes only clockwise. ◇

Now we introduce a successor relation \prec on \mathcal{H}_n , which is related to (\mathcal{E}_n, \prec) defined in 3.2.1, but does not have all the nice properties of a partial order. Reading the following definition, one should have the process $\mathfrak{Z}^{N,n}$ in mind: $\xi \prec \eta$ holds if and only if $\mathfrak{Z}^{N,n}$ can jump from ξ to η in a single step.

Definition 3.4.7 (Successor relation). Let $\xi, \eta \in \mathcal{H}_n$, and suppose that

$$\xi' = \left\{ \xi_{\alpha\beta} = \begin{bmatrix} \hat{\xi}_{\alpha\beta} \\ \check{\xi}_{\alpha\beta} \end{bmatrix} : \alpha \in [a], \beta \in [b_\alpha] \right\}$$

for some integers a and b_α . Define

$$C_{\alpha,\beta} := \left\{ \bullet \in \{\wedge, \vee\} : \dot{\xi}_{\alpha\beta} \neq \emptyset \right\}, \quad (3.29)$$

for each valid combination of α and β . These sets contain the indices of coalescents that have active lineages in the chromosomes $\xi_{\alpha\beta}$ (remember that we have identified the indices 1,2 with symbols \wedge, \vee). For each $\alpha \in [a]$, $\beta \in [b_\alpha]$ and $\bullet \in C_{\alpha,\beta}$, let $\dot{\mu}_{\alpha\beta} \in \mathbb{B}$ be some binary value. For each $\alpha \in [a]$, $\bullet \in \{\wedge, \vee\}$ and $x \in \mathbb{B}$ define an index set:

$$\dot{I}_\alpha(x) := \left\{ \beta \in [b_\alpha] : \dot{\xi}_{\alpha\beta} \neq \emptyset, \quad \dot{\mu}_{\alpha\beta} = x \right\}.$$

3. Coalescents in Fixed Pedigrees

If η can be built from the components of ξ in the following way:

$$\eta = \{\eta_\alpha : \alpha \in [a]\}$$

$$\eta_\alpha = \left\{ \left(\bigcup_{\beta \in \hat{I}_\alpha(x)} \dot{\xi}_{\alpha\beta} \right)_{\bullet \in \{\wedge, \vee\}} : x \in \mathbb{B} \right\}, \quad (3.30)$$

then we say that η is a *successor* of ξ , and write $\xi \prec \eta$.

The following example illustrates this definition on a simple special case.

Example 3.4.8. Let $\xi \in \mathcal{H}_5$ as in the previous example 3.4.5.

Suppose that $a = 2, b_1 = 3, b_2 = 2$, and that the chromosomes in ξ' are numbered as follows:

$$\xi' = \left\{ \begin{bmatrix} \{1, 2\} \\ \emptyset \end{bmatrix}, \begin{bmatrix} \{5\} \\ \{1, 4\} \end{bmatrix}, \begin{bmatrix} \emptyset \\ \{2\} \end{bmatrix}, \begin{bmatrix} \{4\} \\ \emptyset \end{bmatrix}, \begin{bmatrix} \{3\} \\ \{3, 5\} \end{bmatrix} \right\}$$

$$=: \{\xi_{1,1}, \xi_{1,2}, \xi_{1,3}, \xi_{2,1}, \xi_{2,2}\}$$

Each chromosome $\xi_{\alpha\beta} = [\hat{\xi}_{\alpha\beta}, \check{\xi}_{\alpha\beta}]$ contains an active lineage of either the first, or the second, or both coalescents. The sets $C_{\alpha\beta}$ capture this information:

α, β	1, 1	1, 2	1, 3	2, 1	2, 2
$C_{\alpha\beta}$	\wedge	\wedge, \vee	\vee	\wedge	\wedge, \vee

To describe a successor of ξ , we need 7 binary values: $\hat{\mu}_{1,1}, \hat{\mu}_{1,2}, \check{\mu}_{1,2}, \dots$ and so on. More formally: we need $\dot{\mu}_{\alpha\beta} \in \mathbb{B}$ for $\alpha \in [a], \beta \in [b_\alpha], \bullet \in C_{\alpha\beta}$. Suppose that these binary values are given by the following table:

α, β, \bullet	1, 1, \wedge	1, 2, \wedge	1, 2, \vee	1, 3, \vee	2, 1, \wedge	2, 2, \wedge	2, 2, \vee
$\dot{\mu}_{\alpha\beta}$	0	0	1	1	0	0	0

Now we can group those β 's that contribute to different components of different chromosomes:

$$\begin{array}{llll} \hat{I}_1(0) = \{1, 2\} & \hat{I}_1(1) = \emptyset & \hat{I}_2(0) = \{1, 2\} & \hat{I}_2(1) = \emptyset \\ \check{I}_1(0) = \emptyset & \check{I}_1(1) = \{2, 3\} & \check{I}_2(0) = \{2\} & \check{I}_2(1) = \emptyset \end{array}$$

The numbering of elements of ξ' together with these index sets uniquely determine a successor η of ξ :

$$\eta = \left\{ \left\{ \begin{bmatrix} \bigcup_{\beta \in \{1,2\}} \hat{\xi}_{1,\beta} \\ \bigcup_{\beta \in \emptyset} \hat{\xi}_{1,\beta} \end{bmatrix}, \begin{bmatrix} \bigcup_{\beta \in \emptyset} \hat{\xi}_{1,\beta} \\ \bigcup_{\beta \in \{2,3\}} \hat{\xi}_{1,\beta} \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \bigcup_{\beta \in \{1,2\}} \hat{\xi}_{2,\beta} \\ \bigcup_{\beta \in \{2\}} \hat{\xi}_{2,\beta} \end{bmatrix} \right\} \right\}$$

$$= \left\{ \left\{ \begin{bmatrix} \{1, 2, 5\} \\ \emptyset \end{bmatrix}, \begin{bmatrix} \emptyset \\ \{1, 2, 4\} \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \{3, 4\} \\ \{3, 5\} \end{bmatrix} \right\} \right\}.$$

Observe that the two components of $\xi_{1,2}$ ended up in different chromosomes of the same individual: two lineages from two coalescents can end up in the same chromosome, but they tend not to stay together for very long because of the Mendelian randomness.

Furthermore, notice that η would not change if we flipped all binary values for some $\alpha \in [a]$. There are 2^a different choices of $\dot{\mu}_{\alpha\beta}$ that would lead to the same successor η . \diamond

Here are some simple facts about the relationship between the successor relation \prec on \mathcal{H}_n and the partial order (\mathcal{E}_n, \prec) .

Remark 3.4.9. Let $v, w \in \mathcal{H}_n$ and $\xi, \psi, \eta, \theta \in \mathcal{E}_n$ such that

$$\rho(v) = (\xi, \psi), \quad \rho(w) = (\eta, \theta).$$

- 1) $v \prec w$ implies $\xi \prec \eta, \psi \prec \theta$.
- 2) The converse is false in general. Consider the example $\xi, \psi, \eta, \theta = [n]$ and

$$v = \left\{ \left\{ \left[\begin{array}{c} [n] \\ [n] \end{array} \right] \right\} \right\}, \quad w = \left\{ \left\{ \left[\begin{array}{c} [n] \\ \emptyset \end{array} \right] \right\}, \left\{ \left[\begin{array}{c} \emptyset \\ [n] \end{array} \right] \right\} \right\}.$$

Trivially, $\xi \prec \eta$ and $\psi \prec \theta$, but $v \not\prec w$. The reader might rightly object that this degenerate case is not important in the context of coalescents (because it represents a state where both coalescents have reached their MRCA's). However, the same phenomenon occurs also in non-degenerate cases: two active lineages (one from each coalescent) within a single chromosome can eventually separate, but they cannot end up in two distinct individuals after a single step.

- 3) Here is how lineages described in 2) can separate after two steps. It holds: $v \prec u \prec w$ with

$$u = \left\{ \left\{ \left[\begin{array}{c} [n] \\ \emptyset \end{array} \right], \left[\begin{array}{c} \emptyset \\ [n] \end{array} \right] \right\} \right\}.$$

Notice that there are two chromosomes, but they are still in the same individual. In particular, this example shows that the successor relation \prec on \mathcal{H}_n is not transitive.

- 4) The converse of 1) is true if $v \in \text{im}(\iota)$, that is, if $v = \iota(\xi, \psi)$. This means that each active lineage is in its own separate individual. Therefore, we have the freedom to combine these lineages into arbitrary successors $w \in \mathcal{H}_n$, as long as $\rho(w) = (\eta, \theta)$, $\xi \prec \eta$ and $\psi \prec \theta$.

\diamond

3. Coalescents in Fixed Pedigrees

3.5. Functions Φ_a

Before we establish that the process $\mathfrak{Z}^{N,n}$ is indeed a Markov chain and compute the transition probabilities, we have to introduce few more concepts. In particular, we need functions $\Phi_a(b_1, \dots, b_a)$, which should be thought of as probabilities that b_1 lineages hit a certain individual, b_2 lineages hit another individual, and so on, for a different groups of lineages. In the haploid model, similar functions describe the probabilities for a “ b_1, \dots, b_a ”-merger. However, in our case, lineages that hit the same individual do not necessarily merge. We begin with a very simple lemma where we count certain permutations.

Lemma 3.5.1. *Let S be some finite set, k a natural number, and $A_i, B_i \subseteq S$ for each $i \in [k]$ some subsets such that $\{A_i\}_{i=1}^k$ are pairwise disjoint and $\{B_i\}_{i=1}^k$ are also pairwise disjoint. Then it holds:*

$$\#\{\sigma \in \text{Sym}(S) : \bigwedge_{i=1}^k \sigma(A_i) \subseteq B_i\} = (\#S - \sum_{i=1}^k \#A_i)! \cdot \prod_{i=1}^k (\#B_i)_{\#A_i}. \quad (3.31)$$

Proof. Special case. Suppose that $\#A_i = \#B_i$ for all $i \in [k]$, and that both $\{A_i\}_i$ and $\{B_i\}_i$ are coverings of S . Then it holds:

$$\#\{\sigma : \bigwedge_{i=1}^k \sigma(A_i) = B_i\} = \prod_{i=1}^k \# \text{Iso}[A_i, B_i] = \prod_{i=1}^k (\#A_i)!, \quad (3.32)$$

where $\text{Iso}[A_i, B_i]$ is the set of all bijections between A_i and B_i , which has the same cardinality as $\text{Sym}(A_i)$, namely $(\#A_i)!$.

General case. Now let A_i and B_i as in the premise of this lemma. If $\#A_i > \#B_i$ for some i , then both sides of (3.31) are zero. If $\#A_i \leq \#B_i$ for all i , then for each choice of $B'_i \subseteq B_i$ with $\#B'_i = \#A_i$ we can set

$$A_{k+1} := \left(\bigcup_{i=1}^k A_i \right)^c, \quad B'_{k+1} := \left(\bigcup_{i=1}^k B'_i \right)^c$$

to obtain two coverings $\{A_i\}_{i=1}^{k+1}$ and $\{B'_i\}_{i=1}^{k+1}$ as in the special case. Therefore, it holds:

$$\#\{\sigma : \bigwedge_{i=1}^k \sigma(A_i) = B'_i\} = (\#S - \sum_{i=1}^k \#A_i)! \cdot \prod_{i=1}^k (\#A_i)!. \quad (3.33)$$

Summing over all possible choices of $B'_i \in \mathfrak{P}_{\#A_i}(B_i)$ we obtain:

$$\#\{\sigma : \bigwedge_{i=1}^k \sigma(A_i) \subseteq B_i\} = \sum_{B'_1, \dots, B'_k} \#\{\sigma : \bigwedge_{i=1}^k \sigma(A_i) = B'_i\}$$

$$\begin{aligned}
 &= \sum_{B'_1, \dots, B'_k} (\#S - \sum_{i=1}^k \#A_i)! \cdot \prod_{i=1}^k (\#A_i)! \\
 &= \left(\prod_{i=1}^k \binom{\#B_i}{\#A_i} \right) \cdot (\#S - \sum_{i=1}^k \#A_i)! \cdot \prod_{i=1}^k (\#A_i)! \\
 &= (\#S - \sum_{i=1}^k \#A_i)! \cdot \prod_{i=1}^k (\#B_i)_{A_i},
 \end{aligned}$$

thus the equality (3.31) holds. \blacksquare

Corollary 3.5.2. *Let S , A_i , B_i as in the previous lemma, and $\sigma \sim \mathcal{U}_{\text{Sym}(S)}$ a uniformly chosen permutation of S . Then the probability for all A_i 's to end up in corresponding B_i 's after the application of the random permutation σ is given by the following formula:*

$$\mathbb{P} \left[\bigcap_{i=1}^k \{\sigma(A_i) \subseteq B_i\} \right] = \frac{\prod_{i=1}^k (\#B_i)_{\#A_i}}{(\#S)_{\sum_i \#A_i}}.$$

Proof. Divide right hand side of (3.31) by $\#\text{Sym}(S) = (\#S)!$ and apply the definition of the Pochhammer symbol (2.5). \blacksquare

Functions very similar to those in the following definition have been used implicitly by Kingman, but the notation seems to have been introduced by Möhle and Sagitov [8]. The definition is slightly more general than the one commonly used in the context of haploid models, because we allow M and N to be different (with the intent to set $M = 2N$ later).

Definition 3.5.3. Given integers N and M , and exchangeable \mathbb{N}_0 -valued random variables (ν_1, \dots, ν_N) with the property

$$\sum_{i=1}^N \nu_i = M, \tag{3.34}$$

we define for all $a \in \mathbb{N}$ functions $\Psi_a : \mathbb{N}^a \rightarrow [0, 1]$ as follows. Given $b_1, \dots, b_a \in \mathbb{N}$ with $\sum_{\alpha=1}^a b_\alpha \leq M$, find pairwise disjoint sets $B_1, \dots, B_a \subset [M]$ with $\#B_\alpha = b_\alpha$ for all $\alpha \in [a]$, introduce additional randomness by some $\mathcal{U}_{\text{Sym}(M)}$ -distributed random variable σ that is independent of ν and set:

$$\Phi_a(b_1, \dots, b_a) := \mathbb{P} \left[\bigoplus_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \bigcap_{\alpha=1}^a \{\sigma(B_\alpha) \subseteq I(\nu, j_\alpha)\} \right], \tag{3.35}$$

where j_1, \dots, j_a are all pairwise distinct. If $b_1 + \dots + b_a > M$, set $\Phi_a(b_1, \dots, b_a) := 0$.

3. Coalescents in Fixed Pedigrees

In order to ensure that this is well-defined, we have to show that the right hand side of the above expression depends neither on the choice of sets B_α , nor on the random variable σ .

Indeed, with v ranging over all tuples of nonnegative integers (v_1, \dots, v_N) which sum up to M , and with $b := b_1 + \dots + b_a$ it holds:

$$\begin{aligned}
\Phi_a(b_1, \dots, b_a) &= \mathbb{P} \left[\biguplus_v \biguplus_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \bigcap_{\alpha=1}^a \{ \sigma(B_\alpha) \subseteq I(v, j_\alpha) \} \cap \{ \nu = v \} \right] \\
&= \sum_v \sum_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \mathbb{P} \left[\bigcap_{\alpha=1}^a \{ \sigma(B_\alpha) \subseteq I(v, j_\alpha) \} \right] \mathbb{P}[\nu = v] \\
&= \sum_v \sum_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \frac{1}{(M)_b} \prod_{\alpha=1}^a (v_{j_\alpha})_{b_\alpha} \mathbb{P}[\nu = v] \\
&= \sum_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \frac{1}{(M)_b} \mathbb{E} \left[\prod_{\alpha=1}^a (\nu_{j_\alpha})_{b_\alpha} \right] \\
&= \frac{(N)_a}{(M)_b} \mathbb{E} \left[\prod_{\alpha=1}^a (\nu_\alpha)_{b_\alpha} \right], \tag{3.36}
\end{aligned}$$

where we used 3.5.1 in the third line, and exchangeability of the ν_α 's in the last equation. Since the last expression does not contain σ or any B_α 's, the functions Φ_a are well-defined.

If instead of a single underlying variable $\nu = (\nu_1, \dots, \nu_N)$ we have an entire family of variables $\{\nu^i\}_{i \in I}$, we shall make it visible by writing Φ_a^i instead of just Φ_a . The exact meaning should be inferred from the context.

Remark 3.5.4. The explicit formula (3.36) also makes it obvious that all functions Φ_a are symmetric in the sense that their value does not depend on the order of parameters b_1, \dots, b_a . \diamond

Variations of the following simple lemma appear in Möhle's work (see e.g. Lemma 3.1.5 in [7]). However, instead of manipulating combinatoric expressions to prove probabilistic statements, we rather use probabilistic coupling arguments to prove combinatoric identities.

Lemma 3.5.5 (Consistency of Φ). *Let N, M, ν, a and $b_1, \dots, b_a \in \mathbb{N}$ as in the definition 3.5.3. It holds:*

$$\Phi_a(b_1, \dots, b_a) = \sum_{k=1}^a \Phi_a(b_1, \dots, b_{k-1}, b_k + 1, b_{k+1}, \dots, b_a) + \Phi_{a+1}(b_1, \dots, b_a, 1). \tag{3.37}$$

Proof. Let sets B_1, \dots, B_a and permutation σ as in the definition 3.5.3. Let B_{a+1} be yet another set with a single element that is not contained in any other B_α . Fix a realization v of ν . Suppose that there are indices $j_1, \dots, j_a \in [N]$ such that $\sigma(B_\alpha) \subseteq I(v, j_\alpha)$ for all $\alpha \in [a]$. Trivially, $\sigma(B_{a+1})$ is either contained in $I(v, j_k)$ for some $k \in [a]$, or there exists an index $j_{a+1} \in [N]$ distinct from all other indices j_α , such that $\sigma(B_{a+1})$ is contained in $I(v, j_{a+1})$. Thus:

$$\begin{aligned} \bigcap_{\alpha=1}^a \{\sigma(B_\alpha) \subseteq I(v, j_\alpha)\} = \\ \left(\biguplus_{k=1}^a \bigcap_{\substack{\alpha=1 \\ \alpha \neq k}}^a \{\sigma(B_\alpha) \subseteq I(v, j_\alpha)\} \cap \{\sigma(B_k \cup B_{a+1}) \subseteq I(v, j_k)\} \right) \\ \biguplus \left(\bigcup_{\substack{j_{a+1} \\ \text{distinct}}} \bigcap_{\alpha=1}^{a+1} \{\sigma(B_\alpha) \subseteq I(v, j_\alpha)\} \right). \end{aligned}$$

Summing the probabilities of the above events for all possible choices of v and j_1, \dots, j_a yields (3.37). \blacksquare

The following lemma contains estimates similar to those proved by Möhle and Sagitov [8], but our proof is purely measure theoretic, and arguably closer to the intuition.

Lemma 3.5.6 (Anti-monotonicity of Φ). *Let $a, h \in \mathbb{N}$, $b_1, \dots, b_a \in \mathbb{N}$, $g_1, \dots, g_h \in \mathbb{N}$ such that $h \geq a$ and $g_\alpha \geq b_\alpha$ for all $\alpha = 1, \dots, a$. Then it holds:*

$$\Phi_h(g_1, \dots, g_h) \leq \Phi_a(b_1, \dots, b_a). \quad (3.38)$$

Proof. If $g := g_1 + \dots + g_h > M$, then $\Phi_h(g_1, \dots, g_h) = 0$, so there is nothing to show. Otherwise we can find pairwise disjoint subsets G_1, \dots, G_h of $\{1, \dots, M\}$ with $\#G_\chi = g_\chi$ for $\chi = 1, \dots, h$. Then we can choose $B_\alpha \subset G_\alpha$ with $\#B_\alpha = b_\alpha$ for $\alpha = 1, \dots, a$. Clearly, for all permutations $\sigma \in \text{Sym}(M)$ and any subset $X \subseteq \{1, \dots, M\}$, $\sigma(G_\alpha) \subseteq X$ implies $\sigma(B_\alpha) \subseteq X$, and therefore for a random permutation σ it holds:

$$\{\sigma(G_\alpha) \subseteq X\} \subseteq \{\sigma(B_\alpha) \subseteq X\}$$

for all $\alpha = 1, \dots, a$. Now, simply from the monotonicity of measure \mathbb{P} we obtain:

$$\Phi_h(g_1, \dots, g_h) = \mathbb{P} \left[\biguplus_{\substack{j_1, \dots, j_h=1 \\ \text{distinct}}}^N \bigcap_{\chi=1}^h \{\sigma(G_\chi) \subseteq I(\nu, j_\chi)\} \right]$$

3. Coalescents in Fixed Pedigrees

$$\leq \mathbb{P} \left[\biguplus_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \bigcap_{\alpha=1}^a \{ \sigma(B_\alpha) \subseteq I(\nu, j_\alpha) \} \right] \\ = \Phi_a(b_1, \dots, b_a).$$

Therefore, Φ_a are anti-monotonous. ■

The anti-monotonicity of the functions Φ_a becomes useful as soon as we make additional assumptions about the asymptotic behavior of the pair coalescence probability. In the next two lemmas, we want to investigate the asymptotic behavior of Φ_a , as well as asymptotic behavior relative to the pair coalescence probability.

Lemma 3.5.7 (Asymptotic behavior of Φ_a). *Fix some $a \in \mathbb{N}$. For each $N \in \mathbb{N}$, let $M_N \in \mathbb{N} \geq a$ and $\nu^N = (\nu_1^N, \dots, \nu_N^N)$ with the property $\nu_1^N + \dots + \nu_N^N = M_N$ as in 3.5.3. Suppose that $\Phi_1^N(2)$ converges to 0 as $N \rightarrow \infty$. Then it holds:*

$$\lim_{N \rightarrow \infty} \Phi_a^N(b_1, \dots, b_a) = \begin{cases} 1 & \text{if } b_1 = \dots = b_a = 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.39)$$

Proof. For all positive integers b_1, \dots, b_a with $b_\alpha \geq 2$ for some $\alpha \in [a]$, the anti-monotonicity shown in 3.5.6 and the remark 3.5.4 imply:

$$0 \leq \Phi_a^N(b_1, \dots, b_a) \leq \Phi_1^N(2) \xrightarrow{N \rightarrow \infty} 0,$$

thus the second part of (3.39) holds.

On the other hand, for all fixed natural numbers $b \leq M_N$ it holds:

$$\sum_{\xi \in \mathcal{E}_{[b]}} \Phi_{\#\xi}^N((\#\alpha)_{\alpha \in \xi}) = 1. \quad (3.40)$$

To see why this is true, notice that

$$\biguplus_{\xi \in \mathcal{E}_{[b]}} \biguplus_{\substack{j_\alpha, \alpha \in \xi \\ \text{distinct}}} \bigcap_{\alpha \in \xi} \{ \sigma(\alpha) \subseteq I(\nu^N, j_\alpha) \}$$

is merely a complicated way to express that the permutation σ *somehow* maps $[b]$ into $[M_N]$, which is a trivial event with probability 1 (in the above formula, α 's are subsets of $[b]$, and we use ξ itself as the index set).

This implies (with Δ being the finest possible partition of $[b]$):

$$\Phi_b^N(1, \dots, 1) = \Phi_{\#\Delta}^N((\#\alpha)_{\alpha \in \Delta}) \\ = 1 - \sum_{\xi \in \mathcal{E}_{[b]} \setminus \{\Delta\}} \Phi_{\#\xi}^N((\#\alpha)_{\alpha \in \xi}).$$

By pigeonhole principle, for every ξ on the right hand side there must be an $\alpha \in \xi$ such that $\#\alpha \geq 2$. Therefore, by anti-monotonicity shown in lemma 3.5.6 and by the remark 3.5.4, the right hand side converges to 0, so that the first case in (3.39) also holds. ■

In the proof of the next lemma we closely follow Möhle and Sagitov ([9], 5.5).

Lemma 3.5.8 (Pair coalescence is all that matters). *For each $N \in \mathbb{N}$ let M_N and ν^N as in the previous lemma, and furthermore assume that $M_N \rightarrow \infty$ for $N \rightarrow \infty$. Suppose that $\Phi_1^N(2)$ as well as the quotient*

$$\frac{\Phi_1^N(3)}{\Phi_1^N(2)} \quad (3.41)$$

converge to zero. Then it holds:

$$\lim_{N \rightarrow \infty} \frac{\Phi_a^N(b_1, \dots, b_a)}{\Phi_1^N(2)} = \begin{cases} +\infty & \text{if } b_1 = \dots = b_a = 1 \\ 1 & \text{if } b_\alpha = 2 \text{ for exactly one } \alpha \in [a] \text{ and } 1 \text{ otherwise} \\ 0 & \text{otherwise} \end{cases} \quad (3.42)$$

Proof. We investigate three different cases from the above formula separately. We need the third case before the second one, therefore the order will be 1,3,2 rather than 1,2,3.

Case 1: $b_1 = \dots = b_a = 1$.

Since $\Phi_a^N(1, \dots, 1)$ converges to 1 by lemma 3.5.7, while $\Phi_1^N(2)$ is assumed to converge to 0, the first case in (3.42) is obvious.

Case 3: $b_\alpha \geq 3$ for some α , or $b_\alpha \geq 2$ for at least two different α 's.

Fix an $\varepsilon > 0$. Notice that for large enough $x \in \mathbb{R}_{>0}$, the function $x \mapsto (x)_3$ is increasing. Therefore, for all sufficiently large N , we can apply the Markov's inequality:

$$\mathbb{P} [\nu_1^N > \varepsilon M_N] \leq \frac{\mathbb{E}[(\nu_1^N)_3]}{(\varepsilon M_N)_3},$$

hence

$$\frac{N}{\Phi_1^N(2)} \mathbb{P} [\nu_1^N > \varepsilon M_N] \leq \frac{(M_N)_3}{(\varepsilon M_N)_3} \cdot \frac{N \mathbb{E}[(\nu_1^N)_3]}{(M_N)_3 \Phi_1^N(2)} = \frac{(M_N)_3}{(\varepsilon M_N)_3} \cdot \frac{\Phi_1^N(3)}{\Phi_1^N(2)} \xrightarrow{N \rightarrow \infty} 0. \quad (3.43)$$

By exchangeability of ν_1^N, \dots, ν_N^N , we obtain:

$$\Phi_2^N(2, 2) = \frac{(N)_2}{(M_N)_4} \mathbb{E}[(\nu_1^N)_2 (\nu_2^N)_2] = \frac{1}{(M_N)_4} \sum_{i \neq j} \mathbb{E}[(\nu_i^N)_2 (\nu_j^N)_2],$$

We can split each summand $\mathbb{E}[(\nu_i^N)_2 (\nu_j^N)_2]$ depending on whether $\nu_i^N \leq \varepsilon M_N$ or not, and find upper bounds for both parts separately. In the first case it holds:

$$\sum_{i \neq j} \mathbb{E}[(\nu_i^N)_2 (\nu_j^N)_2 \mathbb{1}_{\{\nu_i^N \leq \varepsilon M_N\}}] \leq \varepsilon M_N \sum_{j=1}^N \mathbb{E}[(\nu_j^N)_2 \sum_{i \neq j} (\nu_i - 1)]$$

3. Coalescents in Fixed Pedigrees

$$\begin{aligned}
&\leq \varepsilon M_N^2 \sum_{j=1}^N \mathbb{E} \left[(\nu_j^N)_2 \right] \\
&\leq \varepsilon M_N^2 N \mathbb{E} \left[(\nu_1^N)_1 \right] \\
&= \varepsilon M_N^2 (M_N)_2 \Phi_1^N(2) \\
&\leq \varepsilon M_N^4 \Phi_1^N(2).
\end{aligned}$$

In the second case we obtain:

$$\begin{aligned}
\sum_{i \neq j} \mathbb{E} \left[(\nu_i^N)_2 (\nu_j^N)_2 \mathbb{1}_{\{\nu_i^N > \varepsilon M_N\}} \right] &\leq M_N^3 \sum_{i,j=1}^N \mathbb{E} \left[\nu_j \mathbb{1}_{\{\nu_i^N > \varepsilon M_N\}} \right] \\
&= M_N^4 N \mathbb{P} \left[\nu_1^N > \varepsilon M_N \right].
\end{aligned}$$

Both estimates together entail:

$$\frac{\Phi_2^N(2, 2)}{\Phi_1^N(2)} \leq \frac{M_N^4}{(M_N)_4} \left(\varepsilon + \frac{N}{\Phi_1^N(2)} \mathbb{P} \left[\nu_1^N > \varepsilon M_N \right] \right).$$

By (3.43), the right hand side converges to ε . Since ε could be chosen arbitrarily small, we obtain the convergence $\Phi_2^N(2, 2)/\Phi_1^N(2) \rightarrow 0$.

Since both quotients $\Phi_1^N(3)/\Phi_1^N(2)$ and $\Phi_2^N(2, 2)/\Phi_1^N(2)$ converge to zero, by the anti-monotonicity shown in 3.5.6 we know that $\Phi_a^N(b_1, \dots, b_a)/\Phi_1^N(2)$ must also converge to 0.

Case 2: $b_\alpha = 2$ for exactly one α , 1 otherwise.

Recall the lemma 3.5.5. It holds:

$$\begin{aligned}
\Phi_{a-1}^N(2, 1, \dots, 1) &= \Phi_{a-1}^N(3, 1, \dots, 1) + \sum_{k=2}^{a-1} \Phi_{a-1}^N(2, 1, \dots, 1, 2, 1, \dots, 1) \\
&\quad + \Phi_a^N(2, 1, \dots, 1).
\end{aligned}$$

From case 3 above, we know that the first two summands in this formula are $\mathfrak{o}(\Phi_1^N(2))$. Reading the above formula from right to left, and applying it $(a - 1)$ times, we obtain:

$$\lim_{N \rightarrow \infty} \frac{\Phi_a^N(2, 1, \dots, 1)}{\Phi_1^N(2)} = \lim_{N \rightarrow \infty} \frac{\Phi_{a-1}^N(2, 1, \dots, 1)}{\Phi_1^N(2)} = \dots = \lim_{N \rightarrow \infty} \frac{\Phi_1^N(2)}{\Phi_1^N(2)} = 1,$$

and the proof is finished. ■

Recall the definition 3.4.7, where we introduced the successor relation. A state $\eta \in \mathcal{H}_n$ is a successor of $\xi \in \mathcal{H}_n$ if and only if there is a positive probability for the process $\mathfrak{Z}^{N,n}$ to jump from ξ to η . The functions Ψ_a enable us to express the transition probabilities of the Markov chain $\mathfrak{Z}^{N,n}$ succinctly.

Lemma 3.5.9 (Transition probabilities of $\mathfrak{Z}^{N,n}$). *The process $\mathfrak{Z}^{N,n}$ is a Markov chain with initial distribution*

$$\mathcal{L}(\mathfrak{Z}_0^{N,n}) = \mathcal{L}(\mathcal{H}(\hat{X}_0^{N,n}, \check{X}_0^{N,n})) \quad (3.44)$$

and transition probabilities given by the matrix $\Pi^{(N,n)}$ with entries

$$\begin{aligned} \Pi_{\xi\eta}^{(N,n)} &:= \mathbb{P} \left[\mathfrak{Z}_{g+1}^{N,n} = \eta \mid \mathfrak{Z}_g^{N,n} = \xi \right] \\ &= \begin{cases} \left(\frac{1}{2}\right)^{\#\rho_1(\xi) + \#\rho_2(\xi) - a} \Phi_a^N(b_1, \dots, b_a) & \text{if } \xi \prec \eta \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \quad (3.45)$$

for $\xi, \eta \in \mathcal{H}_n$. Here a and b_α are as in definition 3.4.7.

Proof. At this stage, we cannot simplify the formula for the initial distribution, (3.44) is just the definition.

Suppose that the event $\{\mathfrak{Z}_g^{N,n} = \xi\}$ occurs, that is, there are some distinct $x_{\alpha\beta} \in [N] \times \mathbb{B}$ such that

$$\hat{X}_g^{N,n}|_{\hat{\xi}_{\alpha\beta}} = \check{X}_g^{N,n}|_{\check{\xi}_{\alpha\beta}} = x_{\alpha\beta}.$$

Conditioned on the event $\{\mathfrak{Z}_g^{N,n} = \xi\}$, the occurrence of the event $\{\mathfrak{Z}_{g+1}^{N,n} = \eta\}$ is equivalent to the fulfillment of the following two conditions:

- 1) There must be a distinct individuals in the generation $(g+1)$, with indices $j_1, \dots, j_a \in [N]$, and for each $\alpha \in [a]$ and $\beta \in [b_\alpha]$ it must hold:

$$\pi_1 \circ \hat{X}_{g+1}^{N,n}|_{\hat{\xi}_{\alpha\beta}} = \pi_1 \circ \check{X}_{g+1}^{N,n}|_{\check{\xi}_{\alpha\beta}} = j_\alpha. \quad (3.46)$$

This means: for each α , all sample indices from $\bigcup_{\beta=1}^{b_\alpha} \hat{\xi}_{\alpha\beta} \subset [n]$ must be assigned by $\hat{X}_{g+1}^{N,n}$ to the individual with index j_α (analogously for $\check{X}_{g+1}^{N,n}$). We denote this event by $G(j_1, \dots, j_a)$:

$$G(j_1, \dots, j_a) := \bigcap_{\alpha=1}^a \bigcap_{\beta=1}^{b_\alpha} \left\{ \pi_1 \circ \hat{X}_{g+1}^{N,n}|_{\hat{\xi}_{\alpha\beta}} = j_\alpha \right\}.$$

Notice that it is irrelevant whether we use $\hat{X}^{N,n}$ or $\check{X}^{N,n}$ in the definition, because $G(j_1, \dots, j_a)$ depends only on the underlying random graph, which is common for both processes.

- 2) The second condition deals with the Mendelian randomness. All the relevant values of \hat{m}_g^N and \check{m}_g^N must coincide with $\dot{\mu}_{\alpha\beta}$ up to simultaneous flips of all $\dot{\mu}_{\alpha\beta}$'s for a fixed $\alpha \in [a]$. More precisely, there must be some Boolean values $w_1, \dots, w_a \in \mathbb{B}$ such that for all $\alpha \in [a]$, $\beta \in [b_\alpha]$, $\bullet \in C_{\alpha\beta}$ it holds:

$$\dot{m}_g^N(x_{\alpha\beta}) = \dot{\mu}_{\alpha\beta} \vee w_\alpha,$$

3. Coalescents in Fixed Pedigrees

where \vee denotes the binary XOR operation on Booleans. Let's denote this event as follows:

$$M(w_1, \dots, w_a) := \bigcap_{\alpha=1}^a \bigcap_{\beta=1}^{b_\alpha} \bigcap_{\bullet \in C_{\alpha\beta}} \{ \dot{m}_g^N(x_{\alpha\beta}) = \dot{\mu}_{\alpha\beta} \vee w_\alpha \}.$$

Recall the formula (3.2), by which we defined $X^{N,n}$. It allows us to express the events $G(j_1, \dots, j_a)$ and $M(w_1, \dots, w_a)$ in terms of the random permutation σ_g^N , family sizes ν_g^N , and Boolean random variables \hat{m}_g^N and \check{m}_g^N .

The event $G(j_1, \dots, j_a)$ occurs if and only if for each $\alpha \in [a]$, the permutation σ_g^N maps all $x_{\alpha\beta}$'s into the interval $I(\nu_g^N, j_\alpha)$ (recall that this is a set of ν_{g,j_α}^N contiguous integers):

$$G(j_1, \dots, j_a) = \bigcap_{\alpha=1}^a \left\{ \sigma_g^N(r(\{x_{\alpha\beta}\}_{\beta=1}^{b_\alpha})) \subseteq I(\nu_g^N, j_\alpha) \right\}.$$

We have used events of this sort in the definition 3.5.3, therefore:

$$\sum_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \mathbb{P}[G(j_1, \dots, j_a)] = \Phi_a^N(b_1, \dots, b_a).$$

Recall that \hat{m}_g^N and \check{m}_g^N are just arrays of independent $\text{Ber}(1/2)$ -distributed random variables. The total number of relevant entries can be calculated as follows:

$$\sum_{\alpha=1}^a \sum_{\beta=1}^{b_\alpha} \#C_{\alpha\beta} = \#\rho_1(\xi) + \#\rho_2(\xi),$$

where $C_{\alpha\beta}$ are as in (3.29). Thus we get:

$$\mathbb{P}[M(w_1, \dots, w_a)] = \left(\frac{1}{2}\right)^{\#\rho_1(\xi) + \#\rho_2(\xi)}.$$

Noticing that the events $G(j_1, \dots, j_a)$, $M(w_1, \dots, w_a)$ and $\mathfrak{Z}_g^{N,n} = \xi$ are all independent, we obtain:

$$\begin{aligned} & \mathbb{P}[\mathfrak{Z}_{g+1}^{N,n} = \eta \mid \mathfrak{Z}_g^{N,n} = \xi] \\ &= \sum_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \sum_{w_1, \dots, w_a \in \mathbb{B}} \mathbb{P}[G(j_1, \dots, j_a) \cap M(w_1, \dots, w_a) \mid \mathfrak{Z}_g^{N,n} = \xi] \\ &= \left(\sum_{w_1, \dots, w_a \in \mathbb{B}} \mathbb{P}[M(w_1, \dots, w_a)] \right) \cdot \left(\sum_{\substack{j_1, \dots, j_a=1 \\ \text{distinct}}}^N \mathbb{P}[G(j_1, \dots, j_a)] \right) \end{aligned}$$

3.6. Limiting behavior of two coalescents on common graph

$$= \left(\frac{1}{2}\right)^{\#\rho_1(\xi) + \#\rho_2(\xi) - a} \Phi_a^N(b_1, \dots, b_a).$$

Also notice that the choice of $x_{\alpha\beta}$'s was irrelevant. It means that the probability of the event $\{\mathfrak{Z}_{g+1}^{N,n} = \eta\}$ depends only on $\{\mathfrak{Z}_g^{N,n} = \xi\}$, and not on some hidden values of the underlying processes $\hat{X}^{N,n}$ and $\check{X}^{N,n}$. Thus, $\mathfrak{Z}^{N,n}$ is indeed a Markov chain. ■

3.6. Limiting behavior of two coalescents on common graph

Both the following lemma as well as the subsequent theorem have been proved by Möhle [6]. The lemma is cited in slightly reduced form, the proof is omitted.

Lemma 3.6.1 (Möhle, 1998). *For some dimension $d \in \mathbb{N}$, let $A \in \mathbb{R}^{d \times d}$ be a matrix with*

$$\|A\| := \max_r \sum_{c=1}^d |A_{rc}| = 1,$$

let $(c_N)_{N \in \mathbb{N}_0}$ be a sequence of positive real numbers with $\lim_{N \rightarrow \infty} c_N = 0$. Suppose that $P := \lim_{m \rightarrow \infty} A^m$ exists. Let $(B_N)_{N \in \mathbb{N}_0}$ be a sequence of $d \times d$ matrices such that

$$G := \lim_{N \rightarrow \infty} P B_N P$$

exists. Then for each $t \in [0, \infty)$ it holds:

$$\lim_{N \rightarrow \infty} (A + c_N B_N)^{\lfloor t/c_N \rfloor} = P - I + e^{tG}. \quad (3.47)$$

Proof. Möhle 1998 [6], Lemma 1. ■

The premises of the following theorem (originally proved by Möhle [6]) have been tweaked a little. We removed an unnecessary strict assumption about the sequence $(c_N)_N$. Even though the original proof goes through almost word for word, we include our own interpretation of the proof for completeness.

Theorem 3.6.2 (Separation of time scales). *For each $N \in \mathbb{N}_0$ let $(Y_g^N)_{g \in \mathbb{N}_0}$ be a time-discrete Markov chain with some finite state space E , and let*

$$\Pi^{(N)} := \left(\mathbb{P} [Y_{g+1}^N = \eta \mid Y_g^N = \xi] \right)_{\xi, \eta \in E}$$

be the transition matrix of Y^N . Let $(c_N)_{N \in \mathbb{N}_0}$ be a sequence of positive real numbers that converges to 0. Suppose that the following limits exist:

$$A := \lim_{N \rightarrow \infty} \Pi^N, \quad P := \lim_{m \rightarrow \infty} A^m, \quad G := \lim_{N \rightarrow \infty} P \frac{\Pi^{(N)} - A}{c_N} P.$$

3. Coalescents in Fixed Pedigrees

Furthermore, suppose that the initial distributions $\mathcal{L}(Y_0^N)$ converge weakly to some measure μ on E .

Then the finite dimensional distributions of the processes $(Y_{\lfloor t/c_N \rfloor}^N)_{t \in [0, \infty)}$ converge to those of a time-continuous Markov process $(\mathcal{Y}_t)_t$ with initial distribution μ , transition matrix

$$\Pi(t) = P e^{tG}$$

and infinitesimal generator G .

Proof. For each $N \in \mathbb{N}_0$, set $B_N := c_N^{-1}(\Pi^{(N)} - A)$. From lemma 3.6.1 it follows:

$$\lim_{N \rightarrow \infty} (\Pi^{(N)})^{\lfloor t/c_N \rfloor} = \lim_{N \rightarrow \infty} (A + c_N B_N)^{\lfloor t/c_N \rfloor} = P e^{tG} = \Pi(t).$$

Hence the finite-dimensional distributions of $Y_{\lfloor \cdot/c_N \rfloor}^N$ converge to those of a time-continuous Markov process \mathcal{Y} with initial distribution μ and transition matrix $\Pi(t)$.

Since P is a projection matrix, it holds $P = P^2$. Hence $PG = G$ and

$$P e^{tG} = P \sum_{k=0}^{\infty} \frac{t^k G^k}{k!} = P + \sum_{k=1}^{\infty} \frac{t^k P G^k}{k!} = P - I + e^{tG}.$$

Therefore, the infinitesimal generator is given by

$$\lim_{t \rightarrow 0+} \frac{\Pi(t) - \Pi(0+)}{t} = \lim_{t \rightarrow 0+} \frac{P - I + e^{tG} - P}{t} = \lim_{t \rightarrow 0+} \frac{e^{tG} - I}{t} = G.$$

■

This theorem, together with lemmas 3.5.7 and 3.5.8 now enables us to investigate the asymptotic behavior of the Markov chain $\mathfrak{Z}^{N,n}$.

Lemma 3.6.3 (The *fdl*-limit of $\mathfrak{Z}^{N,n}$). *The finite dimensional distributions of discrete Markov chains $(\mathfrak{Z}_{\lfloor t/c_N \rfloor}^{N,n})_{t \in [0, \infty)}$ converge to those of a time continuous Markov chain $(\mathcal{Z}_t^n)_{t \in [0, \infty)}$ with values in \mathcal{H}_n as N tends towards infinity. The Markov chain \mathcal{Z}^n has*

$$\mathbb{P}[\mathcal{Z}_0^n = \iota(\Delta, \Delta)] = 1 \tag{3.48}$$

as initial distribution, and the transition matrix

$$\Pi^{(n)}(t) = P e^{tG}, \tag{3.49}$$

where P and G are $\mathcal{H}_n \times \mathcal{H}_n$ -matrices given below. The infinitesimal generator G is defined by the following limit:

$$G := \lim_{N \rightarrow \infty} P \frac{\Pi^{(N,n)} - A}{c_N} P. \tag{3.50}$$

3.6. Limiting behavior of two coalescents on common graph

The matrices A and P are defined as follows (for $\xi, \eta \in \mathcal{H}_n$):

$$A_{\xi\eta} := \begin{cases} (\frac{1}{2})^{\#\rho_1(\xi) + \#\rho_2(\xi) - \#\xi'} & \text{if } \xi \prec \eta \text{ and } \#\eta = \#\xi' \\ 0 & \text{otherwise} \end{cases} \quad (3.51)$$

$$P_{\xi\eta} := \begin{cases} 1 & \text{if } \eta = (\iota \circ \rho)(\xi) \\ 0 & \text{otherwise} \end{cases}. \quad (3.52)$$

In words: A puts every chromosome into a separate individual (and possibly splits chromosomes within individuals), P immediately tears all individuals and chromosomes apart, and puts every active lineage into a separate individual.

Proof. Initial distribution. Recall that $\hat{X}_0^{N,n}$ and $\check{X}_0^{N,n}$ are uniformly chosen injective functions from $[n]$ to $[N] \times \{0\} \simeq [N]$. There are $(N)_n$ injective functions from $[n]$ to $[N]$. Regardless of what $\text{im}(\hat{X}_0^{N,n})$ happens to be, there are $(N-n)_n$ injective functions from $[n]$ to $([N] \times \{0\}) \setminus \text{im}(\hat{X}_0^{N,n})$. Therefore, the chance that images of $\hat{X}_0^{N,n}$ and $\check{X}_0^{N,n}$ do not intersect is:

$$\mathbb{P} \left[\text{im}(\hat{X}_0^{N,n}) \cap \text{im}(\check{X}_0^{N,n}) = \emptyset \right] = \frac{(N-n)_n}{(N)_n} \xrightarrow{N \rightarrow \infty} 1.$$

Hence $\mathcal{L}(\mathfrak{Z}_0^{N,n}) \xrightarrow{N \rightarrow \infty} \delta_{\iota(\Delta, \Delta)}$.

Transition probabilities. Set $A := \lim_{N \rightarrow \infty} \Pi^{(N,n)}$, where $\Pi^{(N,n)}$ is the transition matrix of $\mathfrak{Z}^{N,n}$, described in 3.5.9. From the lemma 3.5.7 we know that $\lim_{N \rightarrow \infty} \Phi_a^{N,2N}(b_1, \dots, b_a)$ is either 0 or 1, and that it is 1 if and only if all b_α 's are equal to 1. All b_α 's being equal to 1 means that each chromosome of ξ picks its own separate parent individual from the previous generation, that is $a = \#\eta = \#\xi'$. Thus, we obtain (3.51).

Now let $P := \lim_{m \rightarrow \infty} A^m$. Before we can calculate the entries of P , we need a better understanding of the matrix A . Here are few simple observations.

i) Suppose that $\xi \in \text{im}(\iota)$. Every active lineage is in its own chromosome, thus

$$\#\rho_1(\xi) + \#\rho_2(\xi) - \#\xi' = 0.$$

Hence $A_{\xi\xi} = 1$ and $A_{\xi\eta} = 0$ for all $\eta \neq \xi$.

ii) Suppose that $\xi \notin \text{im}(\iota)$. Write $\theta := (\iota \circ \rho)(\xi)$ for short. There is an intermediate state $\eta \in \mathcal{H}_n$ such that $\xi \prec \eta \prec \theta$ and $A_{\xi\eta} > 0$, $A_{\eta\theta} > 0$. Such a state η can be constructed from ξ as follows:

- 1) Put each chromosome $c = [\hat{c}, \check{c}] \in \xi'$ into a separate individual
- 2) If both \hat{c} and \check{c} are nonempty, split the chromosome c into two chromosomes $[\hat{c}, \emptyset]$ and $[\emptyset, \check{c}]$ (but keep them within the same individual).

3. Coalescents in Fixed Pedigrees

Formally, this can be expressed as follows:

$$\eta := \left\{ \left\{ \begin{bmatrix} \hat{c} \\ \emptyset \end{bmatrix}, \begin{bmatrix} \emptyset \\ \check{c} \end{bmatrix} \right\} \setminus \left\{ \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix} \right\} \right\}_{c \in \xi'} \quad (3.53)$$

Clearly, $\#\eta = \#\xi'$. The entry $A_{\xi\eta}$ is 2^{-k} , where k is the number of chromosomes c with both \hat{c} and \check{c} nonempty. Furthermore, θ is the only successor of η with $\#\theta = \#\eta'$, therefore $A_{\eta\theta} = 1$. Here is a little example that illustrates the relationship between ξ , η and θ :

$$\begin{aligned} \xi &= \left\{ \left\{ \begin{bmatrix} \{1\} \\ \{1,2\} \end{bmatrix}, \begin{bmatrix} \{2\} \\ \emptyset \end{bmatrix} \right\} \right\} \\ \eta &= \left\{ \left\{ \begin{bmatrix} \{1\} \\ \emptyset \end{bmatrix}, \begin{bmatrix} \emptyset \\ \{1,2\} \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \{2\} \\ \emptyset \end{bmatrix} \right\} \right\} \\ \theta &= \left\{ \left\{ \begin{bmatrix} \{1\} \\ \emptyset \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \emptyset \\ \{1,2\} \end{bmatrix} \right\}, \left\{ \begin{bmatrix} \{2\} \\ \emptyset \end{bmatrix} \right\} \right\} \end{aligned}$$

Since all entries of A are non-negative, together $A_{\xi\eta} > 0$ and $A_{\eta\theta} > 0$ imply $A_{\xi\theta}^2 > 0$.

iii) Finally, observe that $A_{\xi\eta} > 0$ implies $\rho(\eta) = \rho(\xi)$ for all $\xi, \eta \in \mathcal{H}_n$.

Here is a summary of the above statements:

- i) if $\xi \in \text{im}(\iota)$, then $A_{\xi\xi} = 1$,
- ii) for all $\xi \in \mathcal{H}_n$ and $\theta = (\iota \circ \rho)(\xi)$ it holds: $(A^2)_{\xi\theta} > 0$,
- iii) $A_{\xi\eta} > 0$ implies $(\iota \circ \rho)(\eta) = (\iota \circ \rho)(\xi)$.

If we interpret matrix A as a transition matrix of a \mathcal{H}_n -valued Markov chain $(Y_k)_k$, then the above statements translate into following:

- i) the states $\xi \in \text{im}(\iota)$ are absorbing,
- ii) for each $\xi \in \mathcal{H}_n$, an absorbing state $(\iota \circ \rho)(\xi)$ can be reached in two steps,
- iii) from each $\xi \in \mathcal{H}_n$, *at most one* absorbing state is reachable (namely $(\iota \circ \rho)(\xi)$).

Let p be the minimum probability of the event that $(Y_k)_k$, starting at some $\xi \in \mathcal{H}_n$, reaches an absorbing state in two steps:

$$p := \inf_{\xi \in \mathcal{H}_n} (A^2)_{\xi, (\iota \circ \rho)(\xi)}$$

Notice that $p > 0$ by the second statement in the above list (\mathcal{H}_n is finite). Thus, again with $\theta = (\iota \circ \rho)(\xi)$, it holds:

$$\mathbb{P}_\xi [Y_m \neq \theta] \leq (1 - p)^{\lfloor m/2 \rfloor} \xrightarrow{m \rightarrow \infty} 0,$$

3.6. Limiting behavior of two coalescents on common graph

and hence

$$P_{\xi\theta} = \left(\lim_{m \rightarrow \infty} A^m \right)_{\xi\theta} = 1 - \lim_{m \rightarrow \infty} \mathbb{P}_\xi [Y_m \neq \theta] = 1.$$

Thus, the formula (3.52) is also valid.

Application of Möhle's theorem 3.6.2 yields a proof of the lemma. ■

The previous lemma might seem somewhat unsatisfactory, because of the unwieldy transition matrix for which we have only a semi-explicit formula. However, the lemma also tells us that the limit process \mathcal{Z}^n , albeit being formally defined as taking values on the whole space \mathcal{H}_n , spends the entire time in a much simpler subspace that can be identified with $\mathcal{E}_n \times \mathcal{E}_n$. This allows us to cherry-pick only the relevant entries of the transition matrix, and ignore all the transitions from the states in which the chain does not spend any time.

Lemma 3.6.4 (Truncated transition matrix). *For a given sample size n , denote by $\tilde{G}^{(n)}$ the $\mathcal{E}_n^2 \times \mathcal{E}_n^2$ -matrix with entries*

$$\tilde{G}_{(\xi,\psi),(\eta,\theta)}^{(n)} := G_{\iota(\xi,\psi),\iota(\eta,\theta)}, \quad (3.54)$$

where ξ, ψ, η, θ are partitions from \mathcal{E}_n , and G is the $\mathcal{H}_n \times \mathcal{H}_n$ -matrix from the previous lemma.

The matrix $\tilde{G}^{(n)}$ is equal to the Kronecker sum of two Q -matrices of the Kingman's coalescent:

$$\tilde{G}^{(n)} = Q^{(n)} \oplus Q^{(n)}. \quad (3.55)$$

Proof. Fix partitions $\xi, \psi, \eta, \theta \in \mathcal{E}_n$, and write $x := \iota(\xi, \psi)$, $y := \iota(\eta, \theta)$ for short. For the rest of this lemma, set $\tilde{a} := \#\xi + \#\psi - 1$.

Since $P_{qw} = A_{qw} = I_{qw}$ for all $q \in \text{im}(\iota)$, and because $AP = P$, we can drop the first projection matrix and replace AP by the identity matrix I in the formula (3.50), obtaining a slightly shorter formula for the matrix entry in question:

$$\tilde{G}_{(\xi,\psi),(\eta,\theta)}^{(n)} = \lim_{N \rightarrow \infty} c_N^{-1} (\Pi^{(N,n)} P - I)_{xy}. \quad (3.56)$$

If $(\Pi^{(N,n)} P)_{xy} > 0$, then there must exist a state $q \in \mathcal{H}_n$ such that $x \prec q$ and $\rho(q) = (\eta, \theta)$. Thus, by the first part of the remark 3.4.9, whenever we want to show that the entry (3.56) is zero, it is sufficient to consider the cases where $\xi \prec \eta$ and $\psi \prec \theta$.

In the lemma 3.5.8 we have shown that for all $a \in \mathbb{N}$, $b_1, \dots, b_a \in \mathbb{N}$ with $a < \tilde{a}$ and $b_1 + \dots + b_a = \tilde{a} + 1$, it holds:

$$\lim_{N \rightarrow \infty} \frac{\Phi_a(b_1, \dots, b_a)}{\Phi_1(2)} = 0.$$

Therefore $c_N^{-1} \Pi_{xq}^{(N,n)}$ converges to 0 as $N \rightarrow \infty$ for all $q \in \mathcal{H}_n$ with $\#q < \tilde{a}$.

3. Coalescents in Fixed Pedigrees

Now we will compute the entries of $\tilde{G}^{(n)}$ by considering multiple different cases. The following formula will serve us as a task list:

$$\begin{aligned}
& \left(Q^{(n)} \oplus Q^{(n)} \right)_{(\xi, \psi), (\eta, \theta)} \\
&= \left(Q^{(n)} \otimes I + I \otimes Q^{(n)} \right)_{(\xi, \psi), (\eta, \theta)} \\
&= Q_{\xi\eta}^{(n)} I_{\psi\theta} + I_{\xi\eta} Q_{\psi\theta}^{(n)} \\
&= \begin{cases} \text{if } \psi \neq \theta : & \begin{cases} \text{if } \xi \neq \eta : & 0 \\ \text{if } \xi = \eta : & \begin{cases} \text{if } \psi \vdash \theta : & 1 \\ \text{if } \psi \not\vdash \theta : & 0 \end{cases} \end{cases} \\ \text{if } \psi = \theta : & \begin{cases} \text{if } \xi \neq \eta : & \begin{cases} \text{if } \xi \vdash \eta : & 1 \\ \text{if } \xi \not\vdash \eta : & 0 \end{cases} \\ \text{if } \xi = \eta : & -\binom{\#\xi}{2} - \binom{\#\psi}{2} \end{cases} \end{cases} . \end{cases} \quad (3.57)
\end{aligned}$$

Case 1: $\psi \neq \theta, \xi \neq \eta$.

If either $\psi \not\vdash \theta$, or $\xi \not\vdash \eta$, then (3.56) equals 0. Assume $\psi \vdash \theta$ and $\xi \vdash \eta$.

Let $q \in \mathcal{H}_n$ with $x \prec q$. Because $\psi \neq \theta$ and $\xi \neq \eta$, both θ and η must have fewer elements than ψ and ξ respectively. Since it is impossible for q to contain more individuals than there are active lineages, it follows:

$$\#q \leq \#\eta + \#\theta \leq (\#\xi - 1) + (\#\psi - 1) = \tilde{a} - 1 < \tilde{a},$$

therefore $c_N^{-1} \Pi_{xy}^{(N,n)}$ converges to 0.

Since this holds for all choices of q with $x \prec q$, and since the contribution of the identity matrix is also 0, the left hand side of (3.56) is 0.

Case 2: $\xi = \eta, \psi \vdash \theta$.

There is only one $q \in \mathcal{H}_n$ with $\#q \geq \tilde{a}$ such that $\rho(q) = (\eta, \theta)$, namely $q = x$. All other $w \in \mathcal{H}_n \setminus \{q\}$ with $\rho(w) = (\eta, \theta)$ have $\#w < \tilde{a}$, and therefore do not contribute to the result. Thus, the chain of equalities in (3.56) can be continued as follows:

$$\begin{aligned}
\lim_{N \rightarrow \infty} c_N^{-1} (\Pi^{(N,n)} P - I)_{xy} &= \lim_{N \rightarrow \infty} c_N^{-1} \Pi_{xy}^{(N,n)} \\
&= \lim_{N \rightarrow \infty} \frac{\left(\frac{1}{2}\right)^{(\tilde{a}+1)-\tilde{a}} \Phi_{\tilde{a}}(2, 1, \dots, 1)}{\frac{1}{2} \Phi_1(2)} \\
&= 1.
\end{aligned}$$

Here, we used lemma 3.5.8. Notice that there are $\#\psi(\#\psi - 1)/2$ different θ 's with $\psi \vdash \theta$.

Case 3: $\xi = \eta, \psi \neq \theta, \psi \not\vdash \theta$.

If $\psi \not\vdash \theta$, then (3.56) becomes 0, as explained above. Assume $\psi \prec \theta$. Since $\psi \neq \theta$

3.6. Limiting behavior of two coalescents on common graph

and $\psi \not\vdash \theta$, it must hold $\#\theta \leq \#\psi - 2$. Similarly to the first case, for each $q \in \mathcal{H}_n$ with $\rho(q) = (\eta, \theta)$ we obtain the estimate

$$\#q \leq \#\eta + \#\theta \leq \#\xi + (\#\psi - 2) = \tilde{a} - 1 < \tilde{a},$$

therefore the relevant entry of $c_N^{-1} \Pi^{(N,n)} P$ vanishes for $N \rightarrow \infty$, and the entry of the $\tilde{G}^{(n)}$ matrix becomes 0.

Case 4: $\psi = \theta, \xi \vdash \eta$.

This case is analogous to case 2, we again obtain a 1. As in the second case, there are $\#\xi(\#\xi - 1)/2$ different η 's with $\xi \vdash \eta$.

Case 5: $\psi = \theta, \xi \neq \eta, \xi \not\vdash \eta$.

Analogous to case 4, we get a 0.

Case 6: $\xi = \eta, \psi = \theta$.

Since $\Pi^{(N,n)} P$ is a stochastic matrix, it holds:

$$\sum_{q \in \mathcal{H}_n} (\Pi^{(N,n)} P - I)_{xq} = 1 - 1 = 0.$$

We have already computed all other relevant entries in the x -th row. The only non-zero entries are the 1's from the second and the fourth case, therefore it holds:

$$\begin{aligned} \lim_{N \rightarrow \infty} c_N^{-1} (\Pi^{(N,n)} P - I)_{xx} &= - \sum_{\substack{q \in \mathcal{H}_n \\ q \neq x}} \lim_{N \rightarrow \infty} c_N^{-1} (\Pi^{(N,n)} P - I)_{xq} \\ &= - \sum_{\substack{q \in \text{im}(\iota) \\ q \neq x}} \lim_{N \rightarrow \infty} c_N^{-1} (\Pi^{(N,n)} P - I)_{xq} \\ &= - \binom{\#\xi}{2} - \binom{\#\psi}{2}. \end{aligned}$$

The outcomes of the case analysis agree with the formula (3.57) for the Kronecker sum $Q^{(n)} \oplus Q^{(n)}$, thus the proof is finished. ■

Remark 3.6.5. Recall that $P_{qw} > 0$ only if $w \in \text{im}(\iota)$, and note that by definition 3.50 the image of G must be contained in the image of P . Thus, by a simple induction over $k \in \mathbb{N}_0$, we obtain:

$$(G^k)_{\iota(\xi, \psi), \iota(\eta, \theta)} = ((\tilde{G}^{(n)})^k)_{(\xi, \psi), (\eta, \theta)}.$$

This, of course, carries over to the definition of the matrix exponential, so that for all $t \in [0, \infty)$ it holds:

$$\exp(tG)_{\iota(\xi, \psi), \iota(\eta, \theta)} = \exp(t\tilde{G}^{(n)})_{(\xi, \psi), (\eta, \theta)}. \quad (3.58)$$

◇

3. Coalescents in Fixed Pedigrees

Now we can prove the following central proposition.

Proposition 3.6.6. *The finite dimensional distributions of $(\hat{\mathfrak{X}}_{[t/c_N]}^{N,n}, \check{\mathfrak{X}}_{[t/c_N]}^{N,n})_t$ converge to those of $(\hat{\mathcal{K}}_t^n, \check{\mathcal{K}}_t^n)_t$ as $N \rightarrow \infty$, where $\hat{\mathcal{K}}^n$ and $\check{\mathcal{K}}^n$ are two independent copies of the Kingman's coalescent.*

Proof. Fix an integer k and times $t_1, \dots, t_k \in [0, \infty)$. We have to show that

$$(\hat{\mathfrak{X}}_{[t_i/c_N]}^{N,n}, \check{\mathfrak{X}}_{[t_i/c_N]}^{N,n})_{i=1}^k \xrightarrow{N \rightarrow \infty} (\hat{\mathcal{K}}_{t_i}^n, \check{\mathcal{K}}_{t_i}^n)_{i=1}^k.$$

First, let's consider the $\mathcal{E}_n \times \mathcal{E}_n$ -valued process $(\rho(Z_t^n))_t$. Fix some partitions $\xi_1, \dots, \xi_k, \psi_1, \dots, \psi_k \in \mathcal{E}_n$ and write $z_k := \iota(\xi_k, \psi_k)$ for short. Moreover, set $t_0 := 0$ and $\xi_0, \psi_0 := \Delta$. Since for all t_i the Markov chain is almost surely in $\text{im}(\iota)$, from the lemma 3.6.4 and the previous remark we obtain:

$$\begin{aligned} & \mathbb{P} \left[\bigcap_{i=1}^k \{ \rho(Z_{t_i}^n) = (\xi_i, \psi_i) \} \right] = \\ &= \mathbb{P} \left[\bigcap_{i=1}^k \{ Z_{t_i}^n = z_i \} \right] \\ &= \mathbb{P} [Z_0^n = \iota(\Delta, \Delta)] \cdot \prod_{i=1}^k \mathbb{P} [Z_{t_i}^n = z_i \mid Z_{t_{i-1}}^n = z_{i-1}] \\ &= 1 \cdot \prod_{i=1}^k \left(P \exp((t_i - t_{i-1})G) \right)_{z_{i-1}, z_i} \\ &= \prod_{i=1}^k \exp((t_i - t_{i-1})\tilde{G}^{(n)})_{(\xi_{i-1}, \psi_{i-1}), (\xi_i, \psi_i)} \\ &= \prod_{i=1}^k \exp((t_i - t_{i-1})(Q^{(n)} \oplus Q^{(n)}))_{(\xi_{i-1}, \psi_{i-1}), (\xi_i, \psi_i)} \\ &= \prod_{i=1}^k \left(\exp((t_i - t_{i-1})Q^{(n)}) \otimes \exp((t_i - t_{i-1})Q^{(n)}) \right)_{(\xi_{i-1}, \psi_{i-1}), (\xi_i, \psi_i)} \\ &= \prod_{i=1}^k \exp((t_i - t_{i-1})Q^{(n)})_{\xi_{i-1}, \xi_i} \cdot \exp((t_i - t_{i-1})Q^{(n)})_{\psi_{i-1}, \psi_i} \\ &= \prod_{i=1}^k \mathbb{P} [\hat{\mathcal{K}}_{t_i}^n = \xi_i \mid \hat{\mathcal{K}}_{t_{i-1}}^n = \xi_{i-1}] \mathbb{P} [\check{\mathcal{K}}_{t_i}^n = \psi_i \mid \check{\mathcal{K}}_{t_{i-1}}^n = \psi_{i-1}] \\ &= \mathbb{P} \left[\bigcap_{i=1}^k \{ (\hat{\mathcal{K}}_{t_i}^n, \check{\mathcal{K}}_{t_i}^n) = (\xi_i, \psi_i) \} \right], \end{aligned}$$

3.6. Limiting behavior of two coalescents on common graph

therefore the finite dimensional distributions of the process $(\rho(Z_t^n))_t$ are the same as those of $(\hat{\mathcal{K}}^n, \check{\mathcal{K}}^n)$.

Notice that since all involved spaces are discrete, the function ρ , as well as its k -fold cartesian product

$$\rho^{\times k}: \mathcal{H}_n^k \rightarrow (\mathcal{E}_n \times \mathcal{E}_n)^k, \quad \rho^{\times k} := \bigtimes_{i=1}^k \rho$$

are continuous. By the mapping theorem ([1], Thm 2.7), the function $\rho^{\times k}$ respects weak limits. It therefore holds:

$$\begin{aligned} \text{w-lim}_{N \rightarrow \infty} \mathcal{L} \left((\hat{\mathfrak{X}}_{[t_i/c_N]}^{N,n}, \check{\mathfrak{X}}_{[t_i/c_N]}^{N,n})_{i=1}^k \right) &= \text{w-lim}_{N \rightarrow \infty} \mathcal{L} \left((\rho(\mathfrak{Z}_{[t_i/c_N]}^{N,n}))_{i=1}^k \right) \\ &= \text{w-lim}_{N \rightarrow \infty} \mathcal{L} \left(\rho^{\times k}((\mathfrak{Z}_{[t_i/c_N]}^{N,n})_{i=1}^k) \right) \\ &= \mathcal{L} \left(\rho^{\times k}((Z_{t_i}^n)_{i=1}^k) \right) \\ &= \mathcal{L} \left((\rho(Z_{t_i}^n))_{i=1}^k \right) \\ &= \mathcal{L} \left((\hat{\mathcal{K}}_{t_i}^n, \check{\mathcal{K}}_{t_i}^n)_{i=1}^k \right). \end{aligned}$$

Here, we applied definitions 3.2.2, 3.4.3 and the remark 3.4.6 in the first step. The mapping theorem is used in the third step. Finally, we used the above statement about the finite dimensional distributions of $(\rho(Z_t^n))_t$ in the last step. ■

Now we have established that the finite dimensional distributions of two coalescents on the same graph converge to those of two independent Kingman's coalescents. However, originally we wanted to prove the convergence of certain Laplace transforms of the states and holding times representations.

The following lemma will allow us to pass from the convergence of finite dimensional distributions of processes $(\hat{\mathfrak{X}}_{[t/c_N]}^{N,n}, \check{\mathfrak{X}}_{[t/c_N]}^{N,n})_t$ to the weak convergence of the corresponding states and holding times.

Lemma 3.6.7. *Let $\hat{\mathfrak{X}}_{[-/c_N]}^{N,n}$, $\check{\mathfrak{X}}_{[-/c_N]}^{N,n}$, $\hat{\mathcal{K}}^n$ and $\check{\mathcal{K}}^n$ as previously. For each $N \in \mathbb{N}$ denote the states and holding times representations by*

$$(\hat{S}^N, \hat{H}^N) := \Theta((\hat{\mathfrak{X}}_{[t/c_N]}^{N,n})_t), \quad (\check{S}^N, \check{H}^N) := \Theta((\check{\mathfrak{X}}_{[t/c_N]}^{N,n})_t),$$

and moreover, define

$$(\hat{S}^\infty, \hat{H}^\infty) := \Theta(\hat{\mathcal{K}}^n), \quad (\check{S}^\infty, \check{H}^\infty) := \Theta(\check{\mathcal{K}}^n).$$

Then it holds:

$$(\hat{S}^N, \check{S}^N, \hat{H}^N, \check{H}^N) \xrightarrow{N \rightarrow \infty} (\hat{S}^\infty, \check{S}^\infty, \hat{H}^\infty, \check{H}^\infty).$$

3. Coalescents in Fixed Pedigrees

Proof. For this lemma, it is more convenient to consider times \hat{T}_j^N and \check{T}_j^N instead of holding times \hat{H}_j^N and \check{H}_j^N . Since \hat{H}_j^N is defined as difference $\hat{T}_{j-1}^N - \hat{T}_j^N$ (see definition 3.3.1), and \hat{T}_n is always zero, it is enough to show that the weak convergence statement holds for

$$(\hat{S}^N, \hat{T}^N) \equiv (\hat{S}_j^N, \hat{T}_{j-1}^N)_{j=2}^n$$

and analogously defined $(\check{S}^N, \check{T}^N)$. Abbreviate for all $N \in \mathbb{N} \cup \{\infty\}$:

$$V^N := (\hat{S}^N, \check{S}^N, \hat{T}^N, \check{T}^N). \quad (3.59)$$

The idea is to define a semiring \mathcal{A} on the space $\mathcal{E}_n^{2(n-1)} \times [0, \infty)^{2(n-1)}$ such that every open set can be represented as countable union of elements of \mathcal{A} , and then to show that

$$\mathbb{P}[V^\infty \in A] \leq \liminf_{N \rightarrow \infty} \mathbb{P}[V^N \in A] \quad (3.60)$$

holds for all $A \in \mathcal{A}$. We abbreviate $E^N := \{V^N \in A\}$ for all $N \in \mathbb{N} \cup \{\infty\}$.

We will use the family of rectangles aligned to a dyadic grid as \mathcal{A} . For each $r \in \mathbb{N}$ define $G_r := 2^{-r}\mathbb{Z}$, and set $G := \bigcup_r G_r$. Consider the family $\tilde{\mathcal{A}}$ of subsets of $\mathcal{E}_n^{2(n-1)} \times \mathbb{R}^{2(n-1)}$:

$$\tilde{\mathcal{A}} := \left\{ \{(\hat{s}, \check{s})\} \times (\hat{a}, \hat{b}] \times (\check{a}, \check{b}] : \hat{s}, \check{s} \in \mathcal{E}_n^{n-1}, \hat{a}, \hat{b}, \check{a}, \check{b} \in G^{n-1} \right\}.$$

It contains the empty set, it is obviously stable under finite intersections, and the relative complement of two boxes from $\tilde{\mathcal{A}}$ can be represented as a finite union of smaller pairwise disjoint boxes, therefore it is a semiring ([4], Def. 1.9). It is also easy to see that any open set can be filled out by countably many boxes from $\tilde{\mathcal{A}}$.

From the definition of a semiring it is immediately obvious that the trace

$$\mathcal{A} := \left\{ A \cap \mathcal{E}_n^{2(n-1)} \times [0, \infty)^{2(n-1)} : A \in \tilde{\mathcal{A}} \right\}$$

of $\tilde{\mathcal{A}}$ on the subset $\mathcal{E}_n^{2(n-1)} \times [0, \infty)^{2(n-1)}$ is a semiring on this subset. Furthermore, it is compatible with the definition of the trace topology: for every open subset U of $\mathcal{E}_n^{2(n-1)} \times [0, \infty)^{2(n-1)}$, we can find an open subset \tilde{U} of $\mathcal{E}_n^{2(n-1)} \times \mathbb{R}^{2(n-1)}$ such that U is the trace of \tilde{U} . Any decomposition of \tilde{U} into countably many half-open $2(n-1)$ -dimensional intervals from $\tilde{\mathcal{A}}$ induces a decomposition of U into disjoint sets from \mathcal{A} .

Now we have to show that (3.60) holds for all elements of \mathcal{A} . We will investigate only half-open intervals, the proof for the elements on the boundary of $\mathcal{E}_n^{2(n-1)} \times [0, \infty)^{2(n-1)}$ is analogous.

First, notice that if $\#\hat{s}_j \neq j$, then $\mathbb{P}[E^\infty] = 0$, and the inequality (3.60) holds trivially. Same holds for atypical choices of \check{s} . Henceforth, assume that $\#\hat{s}_j = \#\check{s}_j = j$.

3.6. Limiting behavior of two coalescents on common graph

Fix an arbitrarily small $\varepsilon > 0$. For $r \in \mathbb{N}$, consider the event that the distance between any two jumps of $(\hat{\mathcal{K}}^n, \check{\mathcal{K}}^n)$ is greater than 2^{-r} :

$$F_r := \left\{ \min_{s \neq t \in J} > 2^{-r} \right\}, \quad (3.61)$$

(here J denotes the (random) set of times at which the process $(\hat{\mathcal{K}}^n, \check{\mathcal{K}}^n)$ jumps). Choose r so large that the probability of F_r becomes greater than $1 - \varepsilon$.

Now we define events D^N that rely only on finitely many values of the underlying processes, but still allow us to reliably detect events E^N .

Suppose that for each $j \in \{2, \dots, n\}$ we can find $\alpha, \beta \in G_r$ such that all of the following conditions are fulfilled:

- $(\alpha, \beta] \subseteq (\hat{a}_j, \hat{b}_j]$,
- $\hat{\mathfrak{X}}_{[\alpha/c_N]}^{N,n} = \hat{s}_j$,
- $\hat{\mathfrak{X}}_{[\beta/c_N]}^{N,n} = \hat{s}_{j-1}$.

Then we can be sure that the time \hat{T}_{j-1}^N lies in the interval $(\hat{a}_j, \hat{b}_j]$ and that $\hat{S}_j^N = \hat{s}_j$. Analogous statements are valid for \check{T}_j^N and \check{S}_j^N for all $N \in \mathbb{N} \cup \{\infty\}$ (with Kingman's coalescents for $N = \infty$). In other words, whenever the event

$$D^N := \bigcap_{\bullet \in \{\wedge, \vee\}} \bigcap_{j=2}^n \bigcup_{\substack{\alpha, \beta \in G_r \\ (\alpha, \beta] \subseteq (\hat{a}_j, \hat{b}_j]}} \left\{ \hat{\mathfrak{X}}_{[\alpha/c_N]}^{N,n} = \hat{s}_j, \hat{\mathfrak{X}}_{[\beta/c_N]}^{N,n} = \hat{s}_{j-1} \right\} \quad (3.62)$$

(with $\hat{\mathcal{K}}_t^n, \check{\mathcal{K}}_t^n$ instead of $\hat{\mathfrak{X}}_{[t/c_N]}^{N,n}, \check{\mathfrak{X}}_{[t/c_N]}^{N,n}$ for $N = \infty$) occurs, the event E^N also occurs, that is: $D^N \subseteq E^N$ for all $N \in \mathbb{N} \cup \{\infty\}$.

Furthermore, if the event F_r occurs, the distance between any two jumps of the process $(\hat{\mathcal{K}}^n, \check{\mathcal{K}}^n)$ is large enough so that we are guaranteed to be able to find α 's and β 's as above, therefore:

$$\mathbb{P}[F_r \cap E^\infty] = \mathbb{P}[F_r \cap D^\infty].$$

Together with the estimate

$$\mathbb{P}[F_r \cap E^\infty] \geq \mathbb{P}[E^\infty] + \mathbb{P}[F_r] - 1 \geq \mathbb{P}[E^\infty] - \varepsilon$$

this yields:

$$\begin{aligned} \mathbb{P}[E^\infty] &\leq \mathbb{P}[F_r \cap E^\infty] + \varepsilon \\ &= \mathbb{P}[F_r \cap D^\infty] + \varepsilon \\ &\leq \mathbb{P}[D^\infty] + \varepsilon \\ &= \lim_{N \rightarrow \infty} \mathbb{P}[D^N] + \varepsilon \end{aligned}$$

3. Coalescents in Fixed Pedigrees

$$\leq \liminf_{N \rightarrow \infty} \mathbb{P} [E^N] + \varepsilon.$$

Here we have used the *fdd*-convergence proved in 3.6.6 in the next-to-last line. Since this estimate holds for any epsilon, we obtain (3.60).

By a corollary to the portmanteau theorem ([1] Thm. 2.5), we obtain weak convergence of V^N to V^∞ , and the proof is finished. ■

3.7. Limiting behavior of a single coalescent

We now can control the second moment, as discussed right after lemma 3.3.4. As promised, we now return to the calculation of the expected value. Since this is just a simpler version of what we did for the second moment, we omit some details.

Proposition 3.7.1. *Finite dimensional distributions of the processes $\mathfrak{X}_{[-/c_N]}^{N,n}$ converge to those of \mathcal{K}^n as N tends to infinity.*

Proof. First, notice that $\mathfrak{X}_{g+1}^{N,n}$ does not depend on exact individual and chromosome indices which $X_g^{N,n}$ assigns to the sample-indices ¹. For $\mathfrak{X}_{g+1}^{N,n}$, it is also irrelevant whether two active lineages in the generation g are in the same individual or not. Since $\mathfrak{X}_{g+1}^{N,n}$ depends only on $\mathfrak{X}_g^{N,n}$, and not on $X_g^{N,n}$, the process $\mathfrak{X}^{N,n}$ is actually a Markov chain. Denote its transition matrix by $\Pi^{(N,n)}$. Clearly, for all $\xi, \eta \in \mathcal{E}_n$ with $\xi \neq \eta$, it holds: $\Pi_{\xi\eta}^{(N,n)} \in \mathcal{O}(c_N)$, therefore by lemma 3.5.7 it holds:

$$\lim_{N \rightarrow \infty} \Pi^{(N,n)} = I,$$

where I denotes a $\mathcal{E}_n \times \mathcal{E}_n$ identity matrix. Let $k \in \mathbb{N}$ arbitrary, let $t_1, \dots, t_k \in [0, \infty)$ be sorted sequence of times, and $\xi_1, \dots, \xi_k \in \mathcal{E}_n$ some partitions. Set $t_0 := 0$ and $\xi_0 := \Delta$. Recall the following well-known identity for matrix exponentials:

$$\lim_{N \rightarrow \infty} (\Pi^{(N,n)})^{\lfloor t/c_N \rfloor} = \exp \left(t \cdot \lim_{N \rightarrow \infty} \frac{\Pi^{(N,n)} - I}{c_N} \right).$$

By repeated application of the elementary Markov property, we obtain:

$$\lim_{N \rightarrow \infty} \mathbb{P} \left[\bigcap_{i=1}^k \left\{ \mathfrak{X}_{\lfloor t_i/c_N \rfloor}^{N,n} = \xi_i \right\} \right] = \prod_{i=1}^k \exp \left((t_i - t_{i-1}) \cdot \lim_{N \rightarrow \infty} c_N^{-1} (\Pi^{(N,n)} - I)_{\xi_{i-1}, \xi_i} \right),$$

therefore it is sufficient to show that $G := \lim_{N \rightarrow \infty} c_N^{-1} (\Pi^{(N,n)} - I)$ is the same as the Q-matrix of the Kingman's coalescent. Let's consider various constellations of $\xi, \eta \in \mathcal{E}_n$.

¹Recall that realizations of $X_g^{N,n}$ are functions from $[n]$ to $([N] \times \mathbb{B})$, whereas $\mathfrak{X}_g^{N,n}$ is partition-valued.

3.7. Limiting behavior of a single coalescent

Case 1: $\xi \vdash \eta$.

It should be pointed out that, unlike in the haploid Cannings model, there are many different ways to obtain a pair coalescence in the $(g+1)$ -th generation. An extreme example: one could in principle obtain a pair coalescence with k active lineages but only $\lceil (k-1)/2 \rceil$ distinct individuals in the parent generation.

However, the only asymptotically relevant case is when exactly two lineages hit the same individual, and every other lineage stays in a separate individual. In this case, the coalescence probability is $\Phi_a(2, 1, \dots, 1)/2$. From lemma 3.5.8 we know that

$$\lim_{N \rightarrow \infty} \frac{\frac{1}{2} \Phi_a(2, 1, \dots, 1)}{c_N} = 1.$$

In all other cases, the coalescence probability is $\mathfrak{o}(c_N)$, and therefore negligible. We obtain $G_{\xi\eta} = 1$.

Case 2: $\xi \neq \eta, \xi \not\vdash \eta$.

Since there is no way how previously coalesced lineages could separate, if $\xi \not\vdash \eta$, then $G_{\xi\eta}$ must be 0.

Assume that $\xi \prec \eta$. It must hold: $\#\eta \leq \#\xi - 2$. This requires coalescence of more than two lineages. From lemma 3.5.8, we know that the probability $\Pi_{\xi\eta}^{(N,n)}$ of such an event is $\mathfrak{o}(c_N)$, and therefore negligible for $N \rightarrow \infty$. We again obtain $G_{\xi\eta} = 0$.

Case 3: $\xi = \eta$.

Since $\Pi^{(N,n)}$ is stochastic, each row of G has to sum up to 0. There are $\#\xi(\#\xi-1)/2$ different $\theta \in \mathcal{E}_n$ such that the condition $\xi \vdash \theta$ applies, therefore

$$G_{\xi\xi} = -\#\xi(\#\xi-1)/2 = -\binom{\#\xi}{2}.$$

The case analysis shows that $G = Q^{(n)}$, and therefore the proof is finished. ■

The following corollary is completely analogous to the lemma 3.6.7.

Corollary 3.7.2. *The law of $\Theta(\mathfrak{X}_{[-/c_N]}^{N,n})$ converges weakly to the law of $\Theta(\mathcal{K}^n)$ as N tends to infinity.*

Proof. There were two crucial properties that made the proof of the lemma 3.6.7 work:

- The underlying process was in some sense monotonous: the number of active lineages (in both coalescents) was non-increasing. This enabled us to detect events E^N using events D^N , which relied on finitely many values of the underlying process.
- The holding times of the limiting process were almost certainly positive, and there were only finitely many jumps. This enabled us to detect events E^N using D^N with arbitrary high sensitivity.

3. Coalescents in Fixed Pedigrees

Here, again, the number of active lineages $\#\mathfrak{X}_{\lfloor t/c_N \rfloor}^{N,n}$ is non-increasing, and the holding times of the limit process \mathcal{K}^n are almost surely positive. Thus, the proof strategy from 3.6.7 works as previously. ■

3.8. Convergence in Skorokhod space

The goal of this section is to show that weak convergence of the states and holding times representation implies weak convergence in the Skorokhod space.

Fortunately, we get the weak convergence in the Skorokhod space almost for free: if the underlying states and holding times happen to converge weakly, all we have to do is to show that the set of discontinuities of Θ^{-1} (denoted by $D_{\Theta^{-1}}$) is a null set with respect to the limit measure $\mathcal{L}(\Theta(\mathcal{K}^n))$.

Lemma 3.8.1 (Continuity of Θ^{-1}). *For the set $D_{\Theta^{-1}}$ of discontinuities of Θ^{-1} it holds:*

$$D_{\Theta^{-1}} \subseteq \left(\mathcal{E}_n^{n-1} \times (0, \infty)^{n-1} \right)^c,$$

in other words: Θ^{-1} is continuous on $\mathcal{E}_n^{n-1} \times (0, \infty)^{n-1}$.

Proof. First, we should choose a specific metrization of $\mathcal{E}_n^{n-1} \times [0, \infty)^{n-1}$. We use a combination of the discrete metric d_{discr} on \mathcal{E}_n^{n-1}

$$d_{\text{discr}}(x, y) := \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{if } x = y \end{cases}$$

and the $\|-\|_\infty$ -norm on $[0, \infty)^{n-1}$ to define the metric d_\times on the product space as follows:

$$d_\times((x, t), (y, s)) := d_{\text{discr}}(x, y) \vee \|t - s\|_\infty.$$

Now, fix a point $(S, H) \equiv (S_i, H_i)_{i=2}^n \in \mathcal{E}_n^{n-1} \times (0, \infty)^{n-1}$ and an arbitrarily small $\varepsilon > 0$. Define T_k for $k \in [n]$ analogously to the construction in 3.3.1, and moreover, define an additional value T_0 :

$$T_k := \sum_{i=k+1}^n H_i \quad T_0 := \sum_{i=2}^n H_i + 1. \quad (3.63)$$

Denote the minimum grid-width by $h := \min_{i=2}^n H_i$. Choose a positive δ

$$\delta := \frac{1}{2} \wedge \frac{h}{3(n-1)} \wedge \frac{h(1 - e^{-\varepsilon})}{2(n-1)^2}, \quad (3.64)$$

and let $(Q, G) \in \mathcal{E}_n^{n-1} \times [0, \infty)^{n-1}$ be another list of states and holding times such that $d_\times((S, H), (Q, G)) < \delta$.

First, notice that since the distance is smaller than 1, the lists of states S and Q must be equal.

3.8. Convergence in Skorokhod space

Define times V_k analogously to T_k by $V_k := \sum_{i=k+1}^n G_i$, and consider the offsets $c_k := V_k - T_k$ for all $k \in [n]$. It holds:

$$|c_k| = |V_k - T_k| \leq \sum_{i=k+1}^n |H_i - G_i| \leq (n-1) \|H - G\|_\infty,$$

from this together with the choice of δ (3.64) we obtain two estimates:

$$|c_k| < \frac{h}{3}, \quad (3.65)$$

$$|c_k| < \frac{h(1 - e^{-\varepsilon})}{2(n-1)}. \quad (3.66)$$

for each $k \in [n]$.

To prove that $d_{\text{Sk}}(\Theta^{-1}(S, H), \Theta^{-1}(Q, G)) < \varepsilon$, it is sufficient to find a strictly increasing $\lambda \in \Lambda$ (as in definition 2.2.1) with $\gamma(\lambda) < \varepsilon$ and

$$\Theta^{-1}(S, H) = \Theta^{-1}(Q, G) \circ \lambda,$$

because in this case, the integral part in the definition of the Skorokhod metric (2.6) simply vanishes. We can build such a λ by adding little corrections to the identity function. For any three real numbers a, b, c with $a < b < c$, define the general tent function

$$\Lambda^{(a,b,c)}(t) := \begin{cases} 0 & \text{for } t \in (-\infty, a] \\ \frac{t-a}{b-a} & \text{for } t \in (a, b] \\ \frac{c-t}{c-b} & \text{for } t \in (b, c] \\ 0 & \text{for } t \in (c, \infty) \end{cases},$$

and abbreviate $\Lambda^k := \Lambda^{(T_{k+1}, T_k, T_{k-1})}$ for $k \in [n-1]$. Notice that Λ^k are differentiable everywhere except at the finitely many points $\{T_k\}_{k=0}^n$, and the maximum absolute value of the slope is at most h^{-1} . Using these tent functions, we now can construct the function λ as follows:

$$\lambda := \text{Id}_{[0, \infty)} + \sum_{i=1}^{n-1} c_i \Lambda^i.$$

The estimate (3.65) ensures that on each interval between T_k 's the first derivative of λ stays within the range

$$\left[1 - 2 \cdot \frac{h}{3} \cdot h^{-1}, 1 + 2 \cdot \frac{h}{3} \cdot h^{-1}\right] = [1/3, 5/3],$$

so that λ is strictly monotonous, and therefore indeed an element of Λ . The other estimate (3.66) gives us another bound for the deviation of the first derivative from the constant 1 function. For each $\tau \in [0, \infty) \setminus \{T_k\}_{k=0}^n$, it holds:

$$|\lambda'(\tau) - 1| \leq \sum_{k=1}^{n-1} 2|c_k| h^{-1} < 2(n-1)h^{-1} \frac{h(1 - e^{-\varepsilon})}{2(n-1)} = 1 - e^{-\varepsilon},$$

3. Coalescents in Fixed Pedigrees

therefore, for each $a, b \in [0, \infty)$ with $a < b$ we obtain:

$$\left| \frac{\lambda(b) - \lambda(a)}{b - a} - 1 \right| \leq \sup_{\tau \neq T_k} |\lambda'(\tau) - 1| < 1 - e^{-\varepsilon},$$

hence

$$\frac{\lambda(b) - \lambda(a)}{b - a} \in (1 - (1 - e^{-\varepsilon}), 1 + (e^{\varepsilon} - 1)) = (e^{-\varepsilon}, e^{\varepsilon}),$$

and finally

$$\left| \log \frac{\lambda(b) - \lambda(a)}{b - a} \right| < \varepsilon.$$

Since this estimate holds for all a, b with $a < b$, we obtain $\gamma(\lambda) < \varepsilon$. As described above, this implies that the Skorokhod distance is also smaller than ε , thus we get continuity at (S, H) . Since (S, H) could be chosen arbitrarily from $\mathcal{E}_n^{n-1} \times [0, \infty)^{n-1}$, the proof is complete. ■

Corollary 3.8.2 (Weak convergence in $D^+([0, \infty), \mathcal{E}_n)$). *If $(Y_N)_N$ is a sequence of random variables with values in $\mathcal{E}_n^{n-1} \times [0, \infty)^{n-1}$ such that $\mathcal{L}(Y_N)$ converge weakly to $\mathcal{L}(\Theta(\mathcal{K}^n))$ as $N \rightarrow \infty$, then the laws of $\Theta^{-1}(Y_N)$ converge weakly to $\mathcal{L}(\mathcal{K}^n)$.*

Proof. Times between jumps of the Kingman's coalescent are exponentially distributed, and therefore almost surely positive. Consequently, $D_{\Theta^{-1}}$ is a null set with respect to the measure $\mathcal{L}(\Theta(\mathcal{K}^n))$. Thus, the claim follows from the mapping theorem. ■

3.9. Putting it all together

In this section, we combine all the building blocks from the previous sections. Before we prove our central theorem 3.2.5, we interject yet another helper lemma that will spare us some juggling with the lengthy expressions denoting our random measures.

Lemma 3.9.1. *Let E be some finite space, $d \in \mathbb{N}$. Let $(\mu_l)_l$ be a sequence of $\mathcal{M}_1(E \times [0, \infty)^d)$ -valued random variables and μ a measure on $E \times [0, \infty)^d$. Suppose that the following two conditions hold:*

$$\|\mathbb{E}[\text{LT}_{\mu_l}(-, -)] - \text{LT}_{\mu}\|_{\infty} \xrightarrow{l \rightarrow \infty} 0 \quad (3.67)$$

$$\sum_{i=1}^{\infty} \|\text{Var}[\text{LT}_{\mu_l}(-, -)]\|_{\infty} \leq \infty, \quad (3.68)$$

where $\mathbb{E}[f(-, -)]$ denotes the function $(y, \lambda) \mapsto \mathbb{E}[f(y, \lambda)]$ (similarly for Var). Then it holds:

$$\mathbb{P} \left[\mu_l \xrightarrow{l \rightarrow \infty} \mu \right] = 1.$$

3.9. Putting it all together

Proof. By the proposition 2.3.6, it is sufficient to show that the event $\{LT_{\mu_l} \xrightarrow{l \rightarrow \infty} LT_{\mu}\}$ has probability 1. Since Laplace transforms are continuous, it is sufficient to check the pointwise convergence on some dense subset of $E \times [0, \infty)^d$, for example on $A := E \times (\mathbb{Q} \cap [0, \infty))^d$. Fix $(y, \lambda) \in A$. Abbreviate $V_l := LT_{\mu_l}(y, \lambda)$, $V := LT_{\mu}(y, \lambda)$ (notice that V_l are random variables, while V is just a real constant). We can express the event of non-convergence at (y, λ) as follows:

$$\begin{aligned} \left\{V_l \xrightarrow{l \rightarrow \infty} V\right\}^c &= \bigcup_{k \in \mathbb{N}} \bigcap_{n \in \mathbb{N}} \bigcup_{m > n} \left\{|V_l - V| > \frac{1}{k}\right\} \\ &= \bigcup_{k \in \mathbb{N}} \limsup_n \left\{|V_n - V| > \frac{1}{k}\right\}. \end{aligned} \quad (3.69)$$

Fix $k \in \mathbb{N}$. By the assumption, $\mathbb{E}[V_l]$ converges to V , therefore we can find L so large that

$$|\mathbb{E}[V_l] - V| < \frac{1}{2k}$$

for all l beyond L . Thus, from the triangle inequality, we get for all l large enough:

$$|V_l - V| \leq |V_l - \mathbb{E}[V_l]| + |\mathbb{E}[V_l] - V| < |V_l - \mathbb{E}[V_l]| + \frac{1}{2k},$$

and thus

$$\left\{|V_l - V| > \frac{1}{k}\right\} \subseteq \left\{|V_l - \mathbb{E}[V_l]| \geq \frac{1}{2k}\right\}.$$

The probability of the event on the right hand side can be bounded using the Chebyshev inequality ([4] 5.11):

$$\mathbb{P}\left[|V_l - \mathbb{E}[V_l]| \geq \frac{1}{2k}\right] \leq 4k^2 \mathbf{Var}[V_l].$$

From this and from the initial assumption (3.68) about the summability of variances we get the following estimate:

$$\begin{aligned} \sum_{l=1}^{\infty} \mathbb{P}\left[|V_l - V| > \frac{1}{k}\right] &\leq L + \sum_{l=L+1}^{\infty} \mathbb{P}\left[|V_l - \mathbb{E}[V_l]| \geq \frac{1}{2k}\right] \\ &\leq L + 4k^2 \sum_{l=L+1}^{\infty} \mathbf{Var}[V_l] \\ &< \infty. \end{aligned}$$

Application of the Borel-Cantelli lemma ([4], Thm. 2.7) yields

$$\mathbb{P}\left[\limsup_n \left\{|V_n - V| > \frac{1}{k}\right\}\right] = 0,$$

and since countable unions of null sets have probability 0, the event in (3.69) almost never occurs. The argument did not depend on the choice of (y, λ) , therefore the statement is true for all elements of the dense subset A . ■

3. Coalescents in Fixed Pedigrees

Now we can finally prove our main theorem.

Proof of Theorem 3.2.5. For each population size $N \in \mathbb{N}$, consider the two conditionally independent random processes $(\hat{\mathfrak{x}}_{[t/c_N]}^{N,n})_t$ and $(\check{\mathfrak{x}}_{[t/c_N]}^{N,n})_t$ (defined as in section 3.4) on a common graph \mathcal{G}^N . Let

$$(\hat{S}^N, \hat{H}^N) = \Theta \left(\hat{\mathfrak{x}}_{[-/c_N]}^{N,n} \right), \quad (\check{S}^N, \check{H}^N) = \Theta \left(\check{\mathfrak{x}}_{[-/c_N]}^{N,n} \right)$$

be the corresponding random vectors of states and holding times of both processes. In 3.6.6 we have established that the finite dimensional distributions of $(\hat{\mathfrak{x}}_{[t/c_N]}^{N,n}, \check{\mathfrak{x}}_{[t/c_N]}^{N,n})_t$ converge to those of two independent Kingman's coalescents $\hat{\mathcal{K}}^n$ and $\check{\mathcal{K}}^n$. From the lemma 3.6.7, we know that this carries over to the corresponding states and holding times, so that $((\hat{S}^N, \hat{H}^N), (\check{S}^N, \check{H}^N))$ converges weakly to $(\Theta(\hat{\mathcal{K}}^n), \Theta(\check{\mathcal{K}}^n))$. Since the function $((\hat{s}, \hat{h}), (\check{s}, \check{h})) \mapsto ((\hat{s}, \check{s}), \hat{h} + \check{h})$ is continuous, by the mapping theorem, $((\hat{S}^N, \check{S}^N), \hat{H}^N + \check{H}^N)$ also converges weakly. Since weak convergence implies pointwise convergence of Laplace transforms (remark 2.3.2), we obtain:

$$\text{LT}_{\mathcal{L}((\hat{S}^N, \check{S}^N), \hat{H}^N + \check{H}^N)}((y, y), \lambda) \xrightarrow{N \rightarrow \infty} \text{LT}_{\mathcal{L}(\Theta(\hat{\mathcal{K}}^n))}(y, \lambda) \cdot \text{LT}_{\mathcal{L}(\Theta(\check{\mathcal{K}}^n))}(y, \lambda).$$

Plugging this together with the results from 3.7.2 into the lemma 3.3.4, we obtain

$$\text{Var} \left[\text{LT}_{\mathcal{L}(\Theta(\mathfrak{x}_{[-/c_N]}^{N,n})|\mathcal{G}^N)}(y, \lambda) \right] \xrightarrow{N \rightarrow \infty} 0 \quad (3.70)$$

for each $(y, \lambda) \in \mathcal{E}_n \times [0, \infty)^d$. Moreover, since all holding times of the Kingman's coalescent are almost surely positive, the corollary 2.3.7 tells us that the above convergence (3.70), as well as the convergence

$$\mathbb{E} \left[\text{LT}_{\mathcal{L}(\Theta(\mathfrak{x}_{[-/c_N]}^{N,n})|\mathcal{G}^N)}(y, \lambda) \right] \xrightarrow{N \rightarrow \infty} \text{LT}_{\mathcal{L}(\Theta(\mathcal{K}^n))}(y, \lambda) \quad (3.71)$$

is uniform in (y, λ) .

Now let $(N_m)_m$ be some strictly increasing sequence of integers. We can thin out this sequence and find a sub-subsequence $(N_{m_l})_l$ such that

$$\sum_{l=0}^{\infty} \left\| \text{Var} \left[\text{LT}_{\mathcal{L}(\Theta(\mathfrak{x}_{[-/c_{N_{m_l}}]}^{N_{m_l},n})|\mathcal{G}^{N_{m_l}})}(-, -) \right] \right\|_{\infty} < \infty$$

becomes summable. By the previous lemma 3.9.1 (with $d = n - 1$, $E = \mathcal{E}_n^{n-1}$, random measures $\mu_l = \mathcal{L}(\Theta(\mathfrak{x}_{[-/c_{N_{m_l}}]}^{N_{m_l},n})|\mathcal{G}^{N_{m_l}})$ and $\mu = \mathcal{L}(\Theta(\mathcal{K}^n))$), it holds:

$$\mathbb{P} \left[\mathcal{L} \left(\Theta \left(\mathfrak{x}_{[-/c_{N_{m_l}}]}^{N_{m_l},n} \right) \middle| \mathcal{G}^{N_{m_l}} \right) \xrightarrow{l \rightarrow \infty} \mathcal{L}(\Theta(\mathcal{K}^n)) \right] = 1.$$

3.9. Putting it all together

Again, because all holding times of the Kingman's coalescent are almost surely positive, it holds (with set of discontinuities $D_{\Theta^{-1}}$ as in 3.8.1):

$$\mathbb{P}[\Theta(\mathcal{K}^n) \in D_{\Theta^{-1}}] = 0.$$

Therefore, by the corollary 3.8.2 we obtain:

$$\mathbb{P}\left[\mathcal{L}\left(\mathfrak{X}_{\lfloor -/c_{N_{m_l}} \rfloor}^{N_{m_l}, n} \middle| \mathcal{G}^{N_{m_l}}\right) \xrightarrow{l \rightarrow \infty} \mathcal{L}(\mathcal{K}^n)\right] = 1$$

To emphasize that this is just the almost sure convergence with respect to the Lévy-Prokhorov metric d_{LP} , we can also state it as follows:

$$\mathbb{P}\left[d_{LP}\left(\mathcal{L}\left(\mathfrak{X}_{\lfloor -/c_{N_{m_l}} \rfloor}^{N_{m_l}, n} \middle| \mathcal{G}^{N_{m_l}}\right), \mathcal{L}(\mathcal{K}^n)\right) \xrightarrow{l \rightarrow \infty} 0\right] = 1.$$

Therefore, for each subsequence we can find an almost surely d_{LP} -convergent sub-subsequence, and the weak limit is always the same, namely $\mathcal{L}(\mathcal{K}^n)$. This is equivalent to convergence in probability (see [4] Cor. 6.13) with respect to the Lévy-Prokhorov metric, thus the theorem holds. ■

4. Simulations

In the previous chapter, we have proved that the laws of coalescents on fixed pedigrees converge stochastically (w.r.t. Lévy-Prokhorov metric) to the Kingman's coalescent. This is a qualitative statement: it tells us that, for large enough population size N , the law of the coalescent on a fixed pedigree probably won't look much different from the standard coalescent. However, it does not tell us anything about the speed of convergence.

In this chapter, we present a simulation framework and experimental results that will give us some rough idea of how quickly the above mentioned laws converge to the Kingman's coalescent. Moreover, we investigate populations with more complex family structures, as well as populations of varying size.

This chapter is structured as follows. In the section 4.1, we briefly describe the framework that we used for simulations. In the section 4.2, we present various family structures that can be represented in our framework. In the last section 4.3, we investigate the influence of varying population size.

4.1. Simulation framework

The basic idea of the experiment is very simple: we generate a random pedigree, sample multiple coalescents within this fixed pedigree, collect some statistics about the sampled coalescents, and then compare the results with what we would expect from the Kingman's coalescent.

The model used for simulations is more general than the model used in the proof. Instead of N individuals per generation, we consider N families per generation. Here, we use the word “family” in the sense of “parental home”, excluding the children (they belong to the next generation). Each of those families can have arbitrarily complex structure, and consist of multiple diploid and haploid individuals of different sexes. The families within the same population can also vary in size. This enables us to model a wide range of family structures, from monogamous couples of diploid individuals (mammals, birds), to colonies of eusocial insects (like ants, bees, wasps).

Generation of a random pedigree can be subdivided into three steps:

1. Sample a sequence $(N_g)_g$, which for each $g \in \mathbb{N}_0$ determines the number of families in the generation g (the number of families can vary over time, but until section 4.3 we assume that it is just a constant N).

4. Simulations

2. For each g , generate a population consisting of N_g families (in some models, there will be more than just one type of family).
3. For each individual, choose a parent family from the previous generation.

Once the random pedigree is generated, we simulate the coalescents using the information about parentship relations from the pedigree, as well as additional source of Mendelian randomness. The exact mechanism of the Mendelian randomness is left abstract, we can easily plug in different implementations for various reproduction mechanisms. The details are somewhat convoluted (this is the main reason why we used a simpler model in our proof), multiple levels of indirection are necessary to keep the mechanism sufficiently general, the interested reader is referred to the source code in the appendix B ¹.

The simulated random coalescents are transformed into states and holding times representation, which then can then be used to collect arbitrary statistics.

We used the object-functional language *Scala* [11] for the implementation. The two most important reasons for this choice were as follows.

First, the OOP-features with a sufficiently expressive type system allow us to implement a generic data structure that is reminiscent of the Giry-monad [3]. This in turn enables us to conveniently compose distributions and to compute certain probabilities exactly, without reverting to sampling.

Second, functional features are helpful when we have to deal with potentially infinite random structures that look like inverse limits of some finite substructures. Since the language does not force us to treat data and algorithms differently, we can easily define random structures that are represented by both sampled data and an algorithm that knows how to generate more data on demand. In particular, this allows us to define potentially infinite random pedigrees. We never specify how many generations we need: if a random coalescent within a pedigree happens to need more generations to reach its MRCA, then the pedigree is extended automatically. Thus, we can avoid some implementation problems described by Wakeley et al. [12], for example, we do not have to make multiple passes through the same finite piece of pedigree if a coalescent turns out to need more steps to converge to the trivial partition.

4.2. Complex family structures

We consider four different models: a reinterpretation of our panmictic diploid model from the previous chapter (we call it “Meme”-model, in a moment we will explain why), human monogamous families (inspired by the “Swedish families” dataset considered by Wakeley et al. [12]), polygynous fish, and colony structures of eusocial insects.

¹ The method `FamilyStructure.chromosomeInheritance()` is responsible for the Mendelian randomness.

4.2.1. Panmictic diploid model as monogamous haploid model

At first glance, the panmictic diploid model used in the proof does not fit into our framework. However, we can simply assign different meaning to certain entities in our panmictic model to obtain an equivalent model with monogamous families of *haploid* individuals. The idea is to reinterpret a diploid individual as a couple consisting of two haploid individuals. Table 4.1 shows the analogy between the two models.

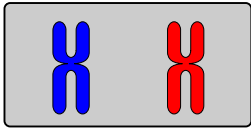
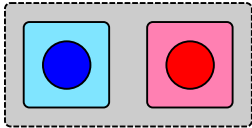
Panmictic diploid model	Monogamous haploid model
Individual with index $i \in [N]$	Couple with index i
	
First chromosome of individual i	Father's meme
Second chromosome of an individual	Mother's meme
First parent chosen at random	Father's parental home chosen at random
Second parent chosen at random	Mother's parental home chosen at random
Number of chromosomes passed to the next generation by i -th individual	Number of children of i -th couple.

Table 4.1.: Analogy between the panmictic haploid model and the monogamous diploid model.

This model does not seem to make much sense in the context of genetics, however, it seems appropriate to describe the propagation of a *meme* through generations. For example, one could think of some idea or technique that each child learns either from his father or from his mother. The Figure 4.1 shows all possible offspring of a family.

We generated 12 different pedigrees for each number of families $N = 10, 50, 100, 1000$, and sampled 10000 coalescents in each pedigree. We plotted the empirical cumulative distribution functions of the holding times H_2 together with the cumulative distribution function of the Exp_1 -law. The Figures 4.3 throughout 4.6 give an impression of how the laws on fixed pedigrees differ from the Exp_1 -law. We can observe that some significant difference is visible for the tiny population size $N = 10$, but already for $N = 50$ the laws on fixed pedigrees are difficult to tell apart from

4. Simulations

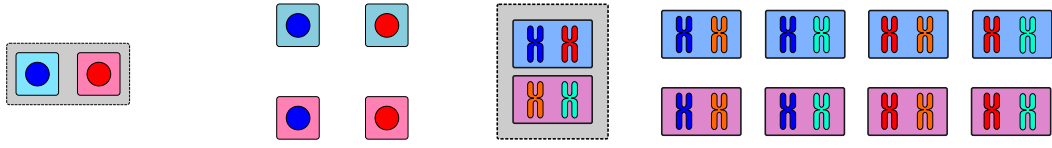


Figure 4.1.: Meme inheritance. On the left, a family with a haploid male and a haploid female is shown. On the right, the top row shows both equiprobable genotypes of male offspring. The bottom row shows two equiprobable genotypes of female offspring.

Figure 4.2.: Mendelian inheritance. On the left, a family with a diploid male and a diploid female is shown. On the right, the top row shows all equiprobable genotypes of male offspring. The bottom row shows all equiprobable genotypes of female offspring.

the Exp_1 -law. However, the error seems to decay rather slowly: the improvements between $N = 50$ and $N = 1000$ are not that obvious.

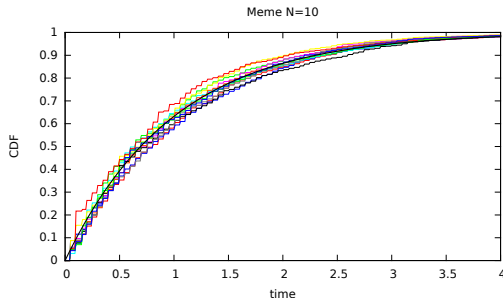


Figure 4.3.: Meme-model, tiny N

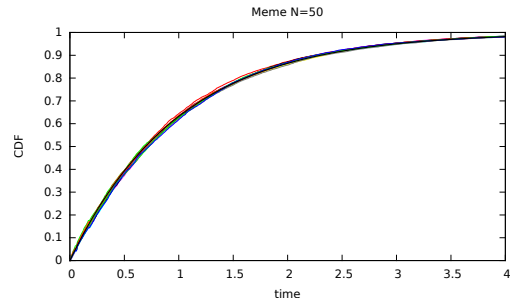


Figure 4.4.: Meme-model, small N

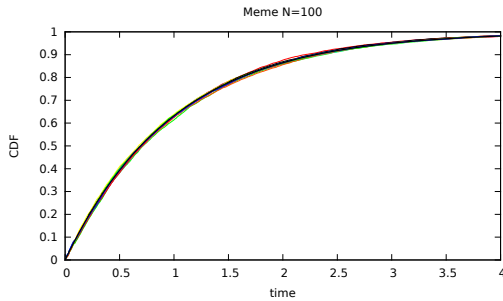


Figure 4.5.: Meme-model, medium N

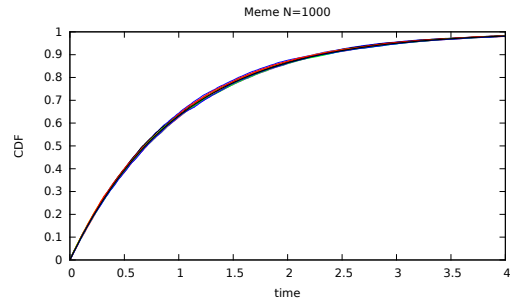


Figure 4.6.: Meme-model, large N

4.2.2. Monogamous families of diploid individuals

Our second model is the most basic model that is applicable to human genetics. We consider disjoint populations of N families, where each family consists of one diploid male and one diploid female. Each individual inherits one chromosome from its father, and one from its mother. Figure 4.2 shows all possible genotypes of the

offspring.

If we consider the position of a lineage within a family, it is clear that the lineage spends roughly one fourth of the time in each chromosome. Thus, the pair coalescence probability c_N is readily computed:

$$c_N = \frac{1}{4} \Phi_1(2).$$

The results of the experiments look very similar to those shown in Figures 4.3 to 4.6, the plots can be found in appendix A.

4.2.3. Polygynous fish

Under the assumption of the Wright-Fisher model for the number of offspring of each couple, the effective population size in the previous two models is just the total number of all chromosomes contained in the population. We wanted to experiment with a model where the effective population size is not trivial.

One such example is provided by certain fish species that live in single-male multiple-female groups [10] (see Figure 4.7). Suppose that N groups (with a varying number of females) inhabit N separate breeding sites. It is reasonable to assume that an observer can track the migration of grown-up individuals (that is, determine which site an adult fish comes from), but cannot determine the mother of a fish (because the tiny eggs are released into water, and cannot be attributed to a unique female). Thus, a fixed pedigree contains only a pointer to the place of birth for every fish. This pointer uniquely determines the father, but the mother has to be chosen uniformly among all females of a group during the coalescent simulation.

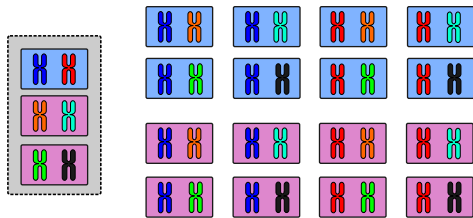


Figure 4.7.: Polygynous fish. On the left: polygynous group with one male and two females. On the right, the top row shows equiprobable genotypes of male offspring. The bottom row shows equiprobable genotypes of female offspring.

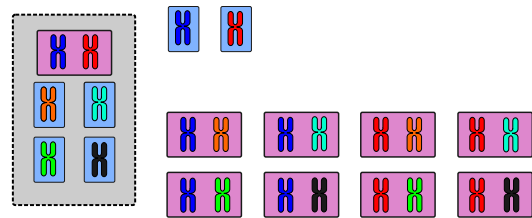


Figure 4.8.: Honeybee colony. On the left: queen and four drones that found a colony. On the right: the top row shows possible genotypes of new drones. The bottom row shows equiprobable genotypes of young queens.

Since every individual inherits one chromosome from the father, and one from the mother, the probability p that a lineage runs through a male fish becomes $1/2$ after a single generation. Since all individuals are diploid, the pair coalescence

4. Simulations

probability is

$$c_N = \left(p^2 \frac{1}{2} + (1-p)^2 \frac{1}{2} \mathbb{E}[F^{-1}] \right) \Phi_1^N(2) = \frac{1}{8} (1 + \mathbb{E}[F^{-1}]) \Phi_1^N(2),$$

where F is the random number of females per breeding site.

For our experiments, we chose the number F of females per site uniformly from $\{1, \dots, 5\}$ (independently for each family), and generated pedigrees for $N = 10, 50, 100$, and 250 . As expected, the results were again similar to those for the Meme-model, see Figures A.5-A.8 in appendix A.

4.2.4. Eusocial insects

In all models considered so far, both males and females inherited their genome in the same way. Eusocial insects (like ants, bees or wasps) provide an example where the inheritance mechanisms for queens (diploid fertile females) and drones (haploid fertile males) are different.

When it's time to found a new colony, queens of the *giant honey bee* (*Apis dorsata*) mate with multiple drones from other colonies [5]. Then they begin to lay eggs. Fertilized eggs develop either into new queens, or infertile workers. Thus, young queens and workers inherit half of their chromosomes from the queen, and half of the chromosomes from one of the drones. Unfertilized eggs develop into male haploid drones, which therefore have to inherit their entire genome from the queen. This is illustrated in the Figure 4.8.

To obtain the correct time scaling, we need the pair coalescence probability. Let p_g denote the probability that a lineage in the generation g goes through a queen. From the above description of the inheritance mechanism, we obtain:

$$p_{g+1} = p_g \frac{1}{2} + (1 - p_g) \cdot 1.$$

This probability rapidly converges to the equilibrium value $p = 2/3$. Thus, the pair coalescence probability is

$$c_N = \left(p^2 \frac{1}{2} + (1-p)^2 \mathbb{E}[D^{-1}] \right) \Phi_1^N(2) = \frac{1}{9} (2 + \mathbb{E}[D^{-1}]) \Phi_1^N(2),$$

where D is the random number of drones that contribute to the genome of a colony.

We conducted our experiments with D chosen uniformly from $\{5, \dots, 10\}$. The results were similar to those of the previous model.

4.3. Varying population size

Constant population size (or rather, constant number of families N), seems to be a rather unrealistic assumption. Therefore, we wanted to find out whether variation in the number of families influences the distribution of MRCA-times.

4.3. Varying population size

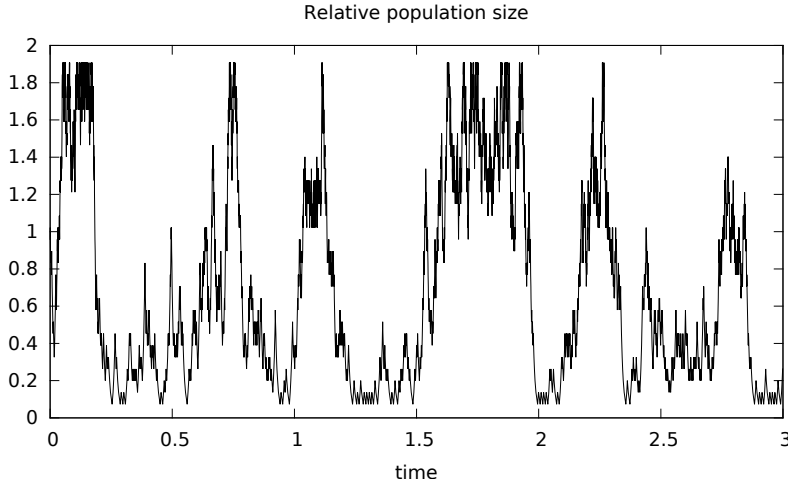


Figure 4.9.: Population size (relative to N), plotted against the intrinsic time. The average number of families is $N = 1000$, relative variation is 0.95, that is, the bounded random walk takes values between 50 and 1950. Notice that the random walk seems squashed at the top, and stretched at the bottom: this is because the time seems to pass faster when the population size is small.

Our implementation allows to make the number of families N time dependent, and to plug in arbitrary time-discrete stochastic processes $(N_g)_g$ instead of the constant function const_N . We use a bounded time-discrete random walk with real-valued increments as our varying population size. The random walk is constrained to the range

$$[N(1 - v), N(1 + v)],$$

where N is the average number of families, and $v \in (0, 1)$ is an additional configurable parameter. The increments of the random walk are scaled with \sqrt{N} in order to keep the relative variance roughly the same for all N .

Of course, we had to rescale the time appropriately: since the number of families varies, the pair coalescence probability does not stay constant either, and thus the intrinsic time does not always run at the same pace. Instead of the processes $(\mathfrak{X}_{[t/c_N]}^{N,n})_t$, we therefore considered processes $(\mathfrak{X}_{\kappa(t)}^{N,n})_t$, where

$$\tau(g) := \sum_{k=1}^g c_{N_g}$$

is an increasing, real-valued, graph-dependent stochastic process that we shall call *the intrinsic time*, and

$$\kappa(t) := \inf \{g \in \mathbb{N}_0 : \tau(g) \geq t\}$$

is a (random) function that distorts the time in such a way that the resulting process seems to “live on the same time-scale” as the standard Kingman’s coalescent.

4. Simulations

The Figure 4.9 shows what the process $(N_g)_g$ can look like.

We repeated our experiments with all previous models with same parameters, but with population size N_g varying between $0.5 \cdot N$ and $1.5 \cdot N$. The results once again confirmed the robustness of the Kingman's coalescent: the ECDF's still looked just like those of the standard Kingman's coalescent. All plots can be found in appendix A.

5. Conclusion

We began by pointing out a discrepancy between the assumptions in the derivation of the Kingman's coalescent, and certain real world problems, for which the Kingman's coalescent is used as a model. In particular, the way Kingman's coalescent is used to describe gene genealogies in fixed pedigrees seemed unjustified.

We reframed the problem as a statement about random coalescents in fixed pedigrees, and formulated our main quenched limit theorem.

The overall strategy was to trade the conditional expectations for a much more complicated Markov chain in a "two times more complicated" state space. This more complicated Markov chain described two coalescents on the same random graph.

These two coalescents occasionally interacted with each other, but always separated quickly. The separation of time scales approach enabled us to separate the short-lived interactions from the actual coalescence events, which took place on a much larger time scale. Then we could prove that for increasingly large populations sizes, the finite dimensional distributions of the two coalescents looked more and more like those of two independent Kingman's coalescents.

This convergence carried over onto the states and holding times representation, which in turn could be transformed into weak convergence in the Skorokhod space. Uniform convergence of Laplace-transforms allowed us to thin out certain sequences of random variables, and show that the weak convergence in the Skorokhod space almost surely occurred for sub-subsequences, which was equivalent to the weak-stochastic convergence, which we used in our theorem.

We have also verified the result by running simulations. Moreover, our flexible simulation framework allowed us to experiment with much more complex family structures and varying population sizes. In all cases, the laws of coalescents on fixed graphs seemed to converge to the law of the Kingman's coalescent.

We conclude that Kingman's coalescent is an appropriate model for describing gene genealogies in fixed pedigrees, as long as there is proper Mendelian randomness, and as long as the underlying pedigree is sufficiently well-behaved.

Appendices

A. Plots

This appendix contains results of the experiments conducted in chapter 4. As one would hope, all plots look essentially the same, confirming the robustness of the Kingman’s coalescent model.

A remark on the nomenclature. During the implementation phase, we called the diploid monogamous model “Duke”, having in mind noble men and women living in a fixed number N of available castles, with a long tradition of writing down their family history. Furthermore, we called our model for eusocial insects “Ants” rather than “Bees”, which would have been more appropriate.

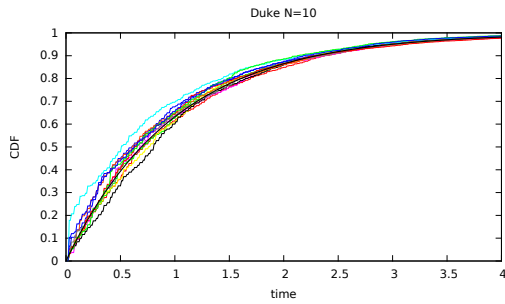


Figure A.1.: Duke, tiny N

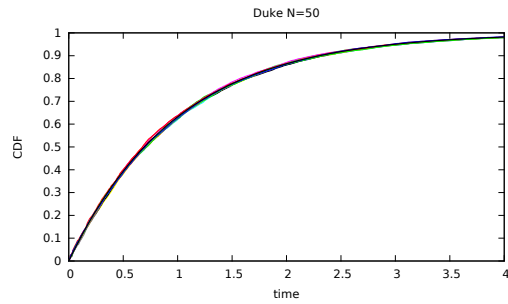


Figure A.2.: Duke, small N

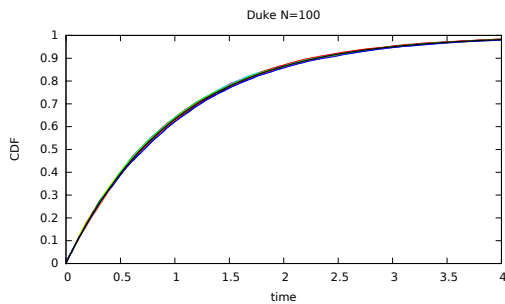


Figure A.3.: Duke, medium N

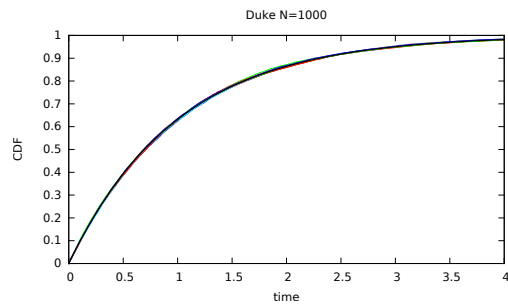


Figure A.4.: Duke, large N

A. Plots

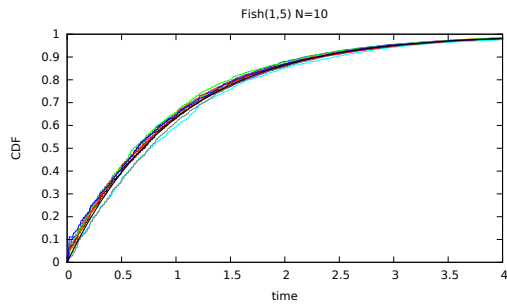


Figure A.5.: Fish(1, 5), tiny N

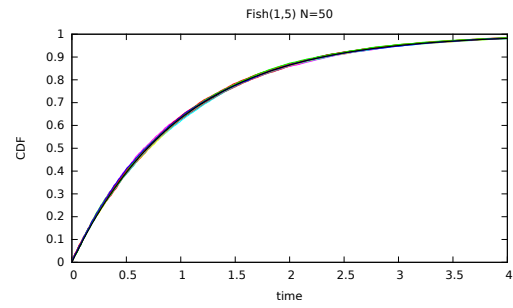


Figure A.6.: Fish(1, 5), small N

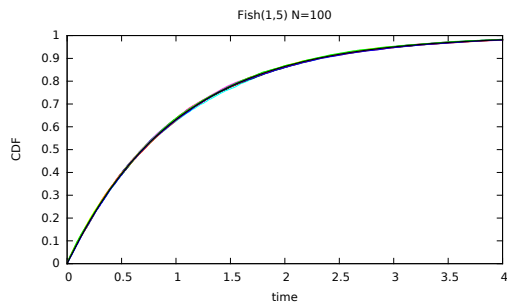


Figure A.7.: Fish(1, 5), medium N

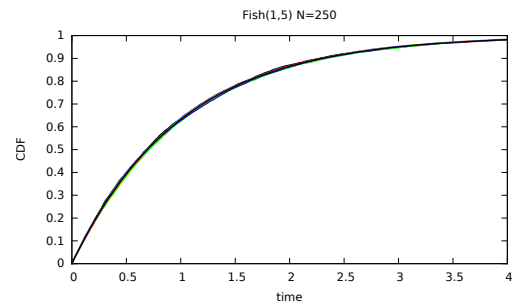


Figure A.8.: Fish(1, 5), large N

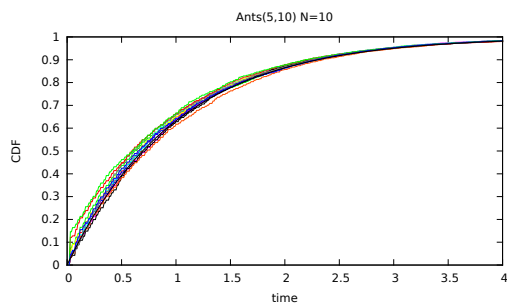


Figure A.9.: Ants(5, 10), tiny N

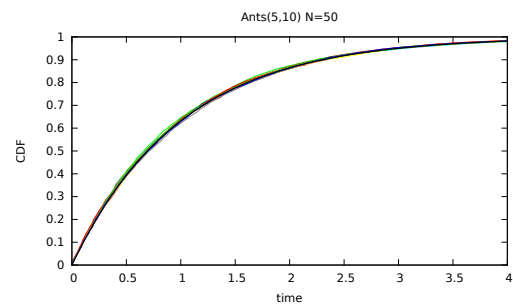


Figure A.10.: Ants(5, 10), small N

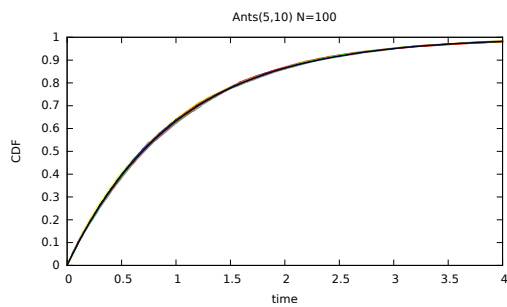


Figure A.11.: Ants(5, 10), medium N

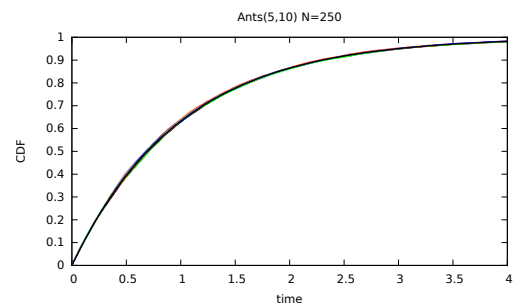


Figure A.12.: Ants(5, 10), large N

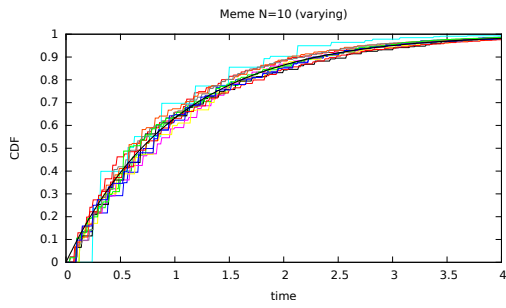


Figure A.13.: Meme, tiny N (varying)

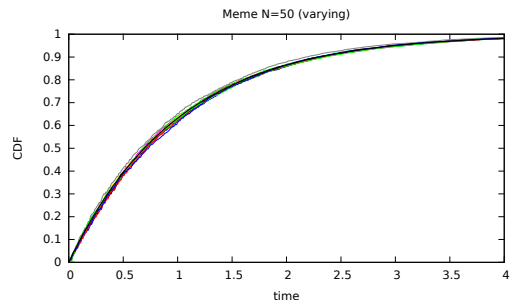


Figure A.14.: Meme, small N (varying)

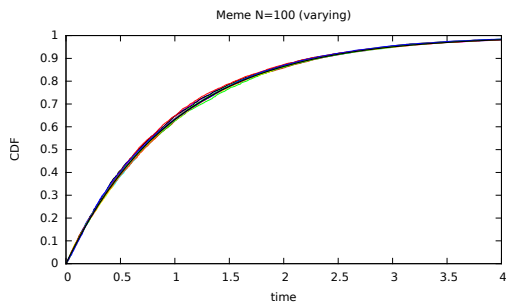


Figure A.15.: Meme, medium N (varying)

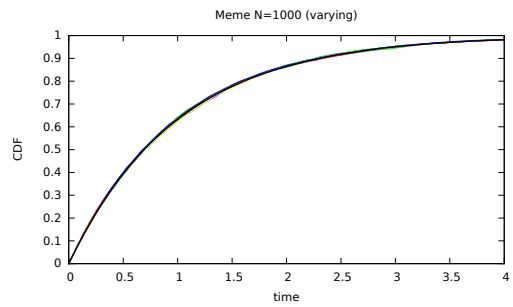


Figure A.16.: Meme, large N (varying)

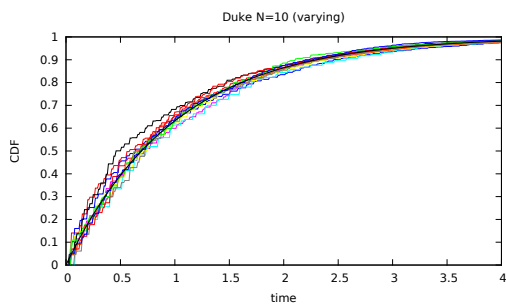


Figure A.17.: Duke, tiny N (varying)

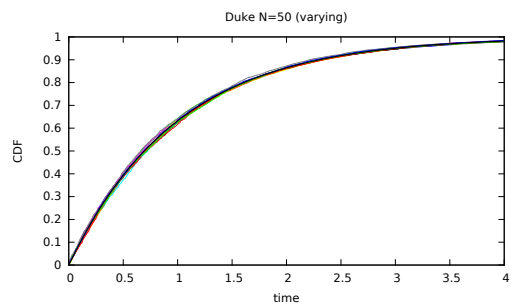


Figure A.18.: Duke, small N (varying)

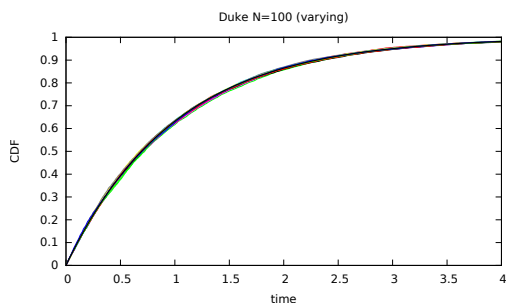


Figure A.19.: Duke, medium N (varying)

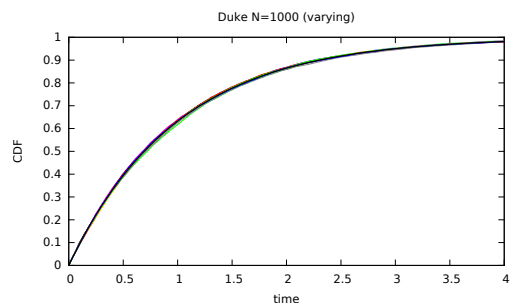


Figure A.20.: Duke, large N (varying)

A. Plots

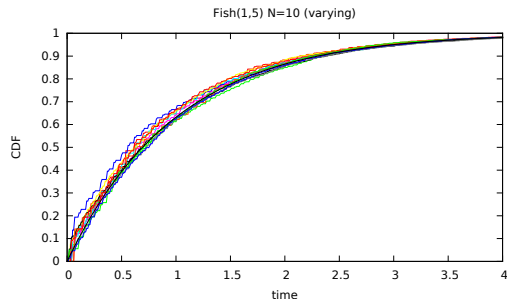


Figure A.21.: Fish(1, 5), tiny N (varying)

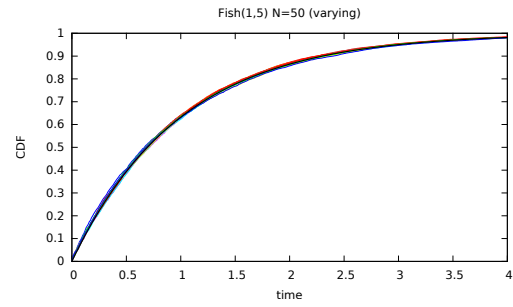


Figure A.22.: Fish(1, 5), small N (varying)

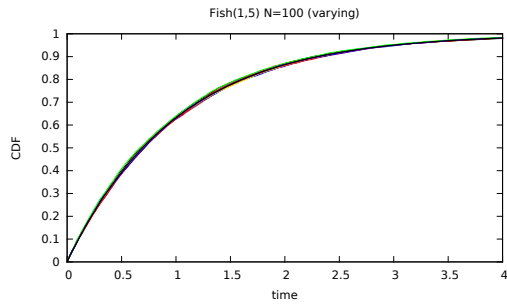


Figure A.23.: Fish(1, 5), medium N (varying)

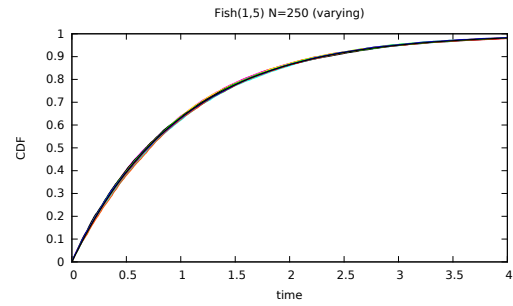


Figure A.24.: Fish(1, 5), large N (varying)

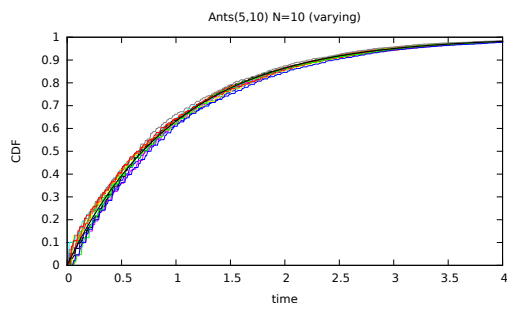


Figure A.25.: Ants(5, 10), tiny N (varying)

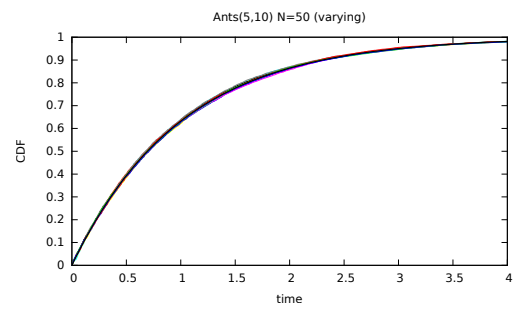


Figure A.26.: Ants(5, 10), small N (varying)

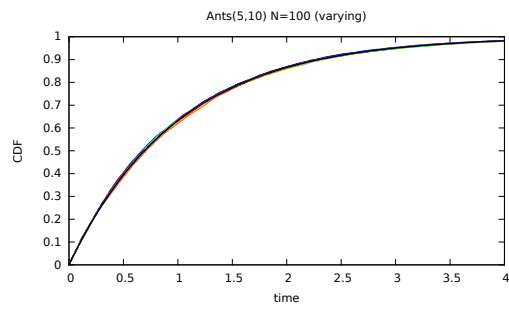


Figure A.27.: $\text{Ants}(5, 10)$, $N \in [50, 150]$

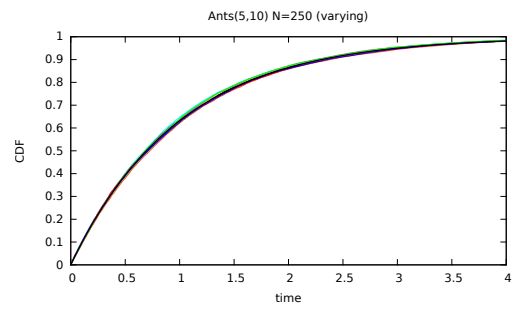


Figure A.28.: $\text{Ants}(5, 10)$, large N (varying)

B. Source code

This appendix includes the entire code that has been used to run the experiments with random coalescents on fixed pedigrees. The language is Scala (version 2.11.2). The code can be git-cloned or downloaded from <https://github.com/tyukiand/coalescentSimulation>.

```
1  /* [INDEX]
2
3  Overview.....25
4  Usage.....47
5  Giry-Monad as 'Distribution' trait.....100
6  Stochastic processes and Markov chains.....482
7  Statistics.....592
8  Partitions.....630
9  Random populations.....668
10 Random pedigrees.....779
11 Coalescents in random pedigrees.....955
12 States and holding times representation.....1103
13 Meme model.....1167
14 'Duke' model (diploid, single locus, one male, one female).....1198
15 Polygynous fish model.....1244
16 Alien-ants model.....1320
17 Code formatting.....1413
18 Parameter parsing.....1476
19 Entry point, running the experiment.....1697
20 Sanity checks for theoretical formulas.....1962
21
22 [/INDEX] */
23
24 /* #####
25  [!] Overview
26  ##### */
27
28 // This software can be used to simulate random coalescents in fixed pedigrees.
29
30 //
31 // The script is organized as follows:
32 // - First, we define some general data structures that are helpful for dealing
33 //   with distributions and stochastic processes
34 // - Then we define a generic model of coalescent in random environment,
35 //   with the underlying model of Mendelian randomness left abstract
36 // - We proceed by defining four concrete family structures
37 // - Then there are some facilities for generating help and formatting code
38 // - Finally, the parameters are parsed, and the requested experiments are run
39
40 /*
41  *
42  * @author Andrey Tyukin
43  * @date 2015-06
44  */
```

B. Source code

```
45
46 /* #####
47 [!] Usage
48 #####*/
49
50 /*
51 * We wanted to minimize the effort that is necessary to get this software
52 * running, and we did so by cramming everything into a single stand-alone
53 * script and avoiding any dependencies.
54 *
55 * This is a stand-alone script that can be executed with the
56 * Scala-interpreter. Assuming that you have Unix/Linux-like environment
57 * with a Scala-interpreter,
58 * all you have to do is to 'cd' into the directory that contains the script,
59 * and issue the following command:
60 * {{{
61 * scala coalescentSimulation.scala --help
62 * }}}
63 * This will display the list of available options and show what a typical
64 * call to this software might look like.
65 *
66 * Here is how one can launch simulations:
67 * {{{
68 * scala coalescentSimulation.scala \
69 * -p 50 -N 100 --num-families-variation 0.8 --model 'Fish(2,5)' \
70 * -c 10000 -n 2 --exp-1-cdf --mrca-ecdf --track-progress --verbose
71 * }}}
72 * The above options mean: simulate 50 different pedigrees with 20-180
73 * fish-families with 2-5 females per family in each generation;
74 * On each pedigree, simulate 10000 coalescents per pedigree with sample size 2
75 * and print the empirical ECDF for each pedigree in the end. Show CDF of Exp_1
76 * for comparison. Track progress, add experiment description to output.
77 *
78 * Less typical application might look as follows:
79 * {{{
80 * scala coalescentSimulation.scala \
81 * -p 1 -N 1000 --num-families-variation 0.9 --model Meme \
82 * --only-populations --verbose
83 * }}}
84 * This would shown only population development, plotted against intrinsic time.
85 *
86 * In case you happen to run out of memory, you have to pass an option to the
87 * JVM used by Scala:
88 * {{{
89 * scala -J-Xmx2048m coalescentSimulation <optionsAsPreviously>
90 * }}}
91 */
92
93 import scala.math._
94 import scala.util.Random
95 import scala.collection.immutable.{Vector}
96 import scala.reflect.ClassTag
97
98
99 /*#####
100 [!] Giry-Monad as 'Distribution' trait
101 #####*/
102
103 /**
104 * Implementation of the Giry-monad.
105 */
106 trait Distribution[X] { outer =>
```

```

107  /** Generates a random realization */
108  def sample: X
109
110  /** Integrates real-valued function `f` exactly */
111  def integral(f: X => Double): Double
112
113  /** Integrates a real-valued function `f` approximately */
114  def approxIntegral(f: X => Double, reps: Int = 1000): Double = {
115    // the default implementation is a very simple Monte-Carlo method
116    var sum = 0d
117    var i = 0
118    while (i < reps) {
119      sum += f(sample)
120      i += 1
121    }
122    sum / reps
123  }
124
125  import Distribution.charFct // defined further below
126
127  /** Computes probability of an event,
128   * this is just integration of characteristic function
129   */
130  def prob(event: X => Boolean): Double = integral(charFct(event))
131
132  def approxProb(event: X => Boolean): Double = integral(charFct(event))
133
134  /** Pushforward probability measure */
135  def map[Y](f: X => Y): Distribution[Y] = new Distribution[Y] {
136    def sample = f(outer.sample)
137    def integral(g: Y => Double) = outer.integral{ x => g(f(x)) }
138  }
139  /** Multiple step random experiment */
140  def flatMap[Y](markovKernel: X => Distribution[Y]): Distribution[Y] =
141  new Distribution[Y] {
142    def sample = markovKernel(outer.sample).sample
143    def integral(f: Y => Double) = outer.integral{
144      x => markovKernel(x).integral(f)
145    }
146  }
147  /** Product with some other, `Y`-valued distribution */
148  def zip[Y](other: Distribution[Y]): Distribution[(X,Y)] =
149  new Distribution[(X,Y)] {
150    def sample = (outer.sample, other.sample)
151    // Fubini
152    def integral(f: ((X, Y)) => Double) = outer.integral{
153      x => other.integral{ y => f((x, y)) }
154    }
155  }
156  /** `n`-fold product with itself */
157  def pow(n: Int): Distribution[Vector[X]] = new Distribution[Vector[X]] {
158    def sample = {
159      (for (i <- 1 to n) yield outer.sample).toVector
160    }
161    def integral(f: Vector[X] => Double) = {
162      // Iterated fubini
163      def integratePartiallyApplied(
164        dim: Int, pa: Vector[X] => Double
165      ): Double = {
166        if (dim == 0) {
167          // all arguments are already plugged in,
168          // `pa` is a function that takes empty vector and returns a constant

```

B. Source code

```
169         pa(Vector())
170     } else {
171         // plug in one more variable, compute inner integral
172         outer.integral{
173             (x: X) => integratePartiallyApplied(dim - 1, {v => pa(v :+ x)})
174         }
175     }
176 }
177 integratePartiallyApplied(n, f)
178 }
179 }
180
181 /** Infinite repetition of the same experiment */
182 def repeat: StochasticProcess[X] = new StochasticProcess[X] {
183     def sample: Stream[X] = outer.sample #:: sample
184     /**
185      * Strangely enough, this actually works, but only as long as
186      * 'f' is guaranteed to look only at finitely many values.
187      * If it looks only "almost surely" at finitely many values, the
188      * method does not terminate.
189      */
190     def integral(f: Stream[X] => Double): Double = {
191         (for {
192             head <- outer
193             tail <- outer.repeat
194         } yield head #:: tail).integral(f)
195     }
196 }
197
198 /**
199  * This distribution conditioned on occurrence of an event of
200  * positive probability.
201  *
202  * Can get very slow if the probability of 'event' is low.
203  */
204 def filter(event: X => Boolean): Distribution[X] = new Distribution[X] {
205     def sample = {
206         val proposal = outer.sample
207         if (event(proposal)) proposal else sample
208     }
209     val eventProbability = prob(event)
210     def integral(f: X => Double) = outer.integral{
211         x => f(x) * charFct(event)(x)
212     } / eventProbability
213 }
214 }
215
216 object Distribution {
217     /** Just a formality to make the definition of the Giry-monad complete */
218     def unit[X](x: X) = Dirac(x)
219
220     /** Transforms predicates into characteristic functions */
221     def charFct[X](event: X => Boolean): (X => Double) = {
222         x => if (event(x)) 1.0 else 0.0
223     }
224 }
225
226 /** Dirac measure (assigns probability '1' to a single outcome) */
227 case class Dirac[X](constant: X) extends Distribution[X] {
228     def sample = constant
229     def integral(f: X => Double) = f(constant)
230 }
```

```

231
232 /** Coin flip with two outcomes, `true` or `false` */
233 case class Bernoulli(p: Double = 0.5) extends Distribution[Boolean] {
234   private val rnd = new Random
235   def sample = rnd.nextDouble < p
236   def integral(f: Boolean => Double) = p * f(true) + (1-p) * f(false)
237 }
238
239 /** Same as mapped `Bernoulli` */
240 case class GenBernoulli[X](t: X, f: X, p: Double = 0.5) extends Distribution[X]{
241   private val rnd = new Random
242   def sample = if (rnd.nextDouble < p) t else f
243   def integral(g: X => Double) = p * g(t) + (1-p) * g(f)
244 }
245
246 /** Uniform distribution on intervals of integers */
247 case class IntUniform(minIncl: Int, maxExcl: Int) extends Distribution[Int] {
248   private val size = maxExcl - minIncl
249   private val rnd = new Random
250   def sample = minIncl + rnd.nextInt(size)
251   def integral(f: Int => Double) =
252     (for (i <- minIncl until maxExcl) yield f(i)).sum / size
253 }
254
255 case class RealUniform(min: Double, max: Double) extends Distribution[Double] {
256   private val rnd = new Random
257   private val diff = max - min
258   def sample = min + rnd.nextDouble * diff
259   def integral(f: Double => Double) = ??? // just ordinary integration
260 }
261
262 /** Uniform distribution on finite sets */
263 case class FiniteUniform[X](values: Array[X]) extends Distribution[X] {
264   private val rnd = new scala.util.Random
265   private val size = values.size
266   def sample = values(rnd.nextInt(size))
267   def integral(f: X => Double) = (for (x <- values) yield f(x)).sum
268 }
269
270 /** Non-uniform distribution on finite sets */
271 class Categorical[X] private (
272   val values: Array[X],
273   val probabilities: Array[Double],
274   val cumulatedProbabilities: Array[Double]
275 ) extends Distribution[X] {
276
277   private val rnd = new Random
278
279   def sample: X = {
280     val i = Categorical.infIndex(cumulatedProbabilities, rnd.nextDouble)
281     values(i)
282   }
283
284   def integral(f: X => Double) = {
285     (for ((v,p) <- values zip probabilities) yield f(v) * p).sum
286   }
287 }
288
289 object Categorical {
290
291   /**
292     * Constructs a finite distribution with given values and weights.

```

B. Source code

```
293     * The weights do not have to sum up to 1.
294     */
295   def apply[X](values: Array[X], weights: Array[Double]): Categorical[X] = {
296     require(
297       !values.isEmpty,
298       "Attempted to construct Categorical distribution on empty set"
299     )
300     val totalWeight = weights.sum
301     require(totalWeight >= 0)
302     require(weights.forall(_ >= 0))
303     for (i <- 0 until weights.size) {
304       weights(i) /= totalWeight
305     }
306     val cumulatedProbabilities =
307       weights.scanLeft(0d){(x, y) => x + y}.tail
308     // artificially add +\infty to the last element
309     cumulatedProbabilities(weights.size-1) += Double.PositiveInfinity
310     new Categorical(values, weights, cumulatedProbabilities)
311   }
312
313   /**
314    * Constructs a finite distribution with given values and probability vector
315    */
316   def apply[X:ClassTag](valuesProbs: Array[(X, Double)]): Categorical[X] = {
317     val (vals, probs) = valuesProbs.unzip
318     this.apply(vals.toArray, probs.toArray)
319   }
320
321   // This part is surprisingly nasty:
322   // finds the smallest index 'i' such that p <= c(i)
323   private[Categorical] def infIndex(c: Array[Double], p: Double): Int = {
324     val bs = java.util.Arrays.binarySearch(c, p)
325     if (bs > 0) {
326       // almost infinitely improbable event, but it can occur on real machine
327       // we have to walk backward until 'c' actually jumps, otherwise we could
328       // return an event of probability 0
329       var i = bs
330       while (c(i) == p && i > 0) i -= 1;
331       i
332     } else if (bs == 0) {
333       0
334     } else {
335       -bs - 1
336     }
337   }
338 }
339
340 case class Permutation(mapping: Array[Int]) extends (Int => Int) {
341   def apply(i: Int) = mapping(i)
342   override def toString = mapping.mkString("(", ", ", ")")
343   def shuffle[A](v: Vector[A]): Vector[A] = {
344     require(mapping.size == v.size)
345     Vector.tabulate(v.size){i => v(mapping(i))}
346   }
347   // exactly the same as above, modulo "JVM-curse": arrays are still aliens...
348   def shuffle[A: ClassTag](arr: Array[A]): Array[A] = {
349     require(mapping.size == arr.size)
350     Array.tabulate(arr.size){i => arr(mapping(i))}
351   }
352 }
353
354 case class UniformPermutation(n: Int) extends Distribution[Permutation] {
```

```

355 private val rnd = new Random
356 def sample: Permutation = {
357     val mapping = Array.tabulate(n){i => i}
358     var tmp: Int = 0
359     for (i <- 0 until n) {
360         val a = rnd.nextInt(n - i)
361         val b = n - 1 - i
362         tmp = mapping(a)
363         mapping(a) = mapping(b)
364         mapping(b) = tmp
365     }
366     Permutation(mapping)
367 }
368 def integral(f: Permutation => Double) = ??? // not that important here
369 }
370
371 import scala.collection.mutable.HashSet
372
373 /**
374  * Generates a random injective function from `{0,...,a-1}` to `{0,...,b-1}`,
375  * represented by an integer array.
376  */
377 case class UniformInjection(a: Int, b: Int)
378 extends Distribution[Array[Int]] {
379     private val rnd = new Random
380
381     private def permutationMethod(a: Int, b: Int): Array[Int] = {
382         val mapping = Array.tabulate(b){i => i}
383         var tmp: Int = 0
384         for (i <- 0 until a) {
385             val x = rnd.nextInt(b - i)
386             val y = b - 1 - i
387             tmp = mapping(x)
388             mapping(x) = mapping(y)
389             mapping(y) = tmp
390         }
391         Array.tabulate(a){i => mapping(b - 1 - i)}
392     }
393
394     private def retryMethod(a: Int, b: Int): Array[Int] = {
395         val chosen = new HashSet[Int]
396         val res = new Array[Int](a)
397         var i = 0
398         while (i < a) {
399             val cand = rnd.nextInt(b)
400             if (!chosen.contains(cand)) {
401                 res(i) = cand
402                 i += 1
403                 chosen += cand
404             }
405         }
406         res
407     }
408
409     def sample = {
410         val C_swap = 7
411         val C_arr = 2
412         val C_hash = 8
413         val retryCost = b * C_hash * math.log(b / (b - a + 1).toDouble)
414         val permutationCost = C_arr * b + C_swap * a
415         if (retryCost < permutationCost){
416             retryMethod(a, b)

```

B. Source code

```
417     } else {
418         permutationMethod(a, b)
419     }
420 }
421
422 def integral(f: Array[Int] => Double) = ???
423 }
424
425 /**
426  * Mixture of finitely many measures is essentially just a two step
427  * experiment: first, we choose a measure, then we sample with respect
428  * to the chosen measure.
429  */
430 class Mixture[X](
431     val components: Array[Distribution[X]],
432     val weights: Array[Double]
433 ) {
434     private val twoStep =
435         for (m <- Categorical(components, weights); x <- m) yield x
436     def sample = twoStep.sample
437     def integral(f: X => Double) = twoStep.integral(f)
438 }
439
440 /**
441  * Empirical distribution on the real number line.
442  *
443  * Essentially a mixture of Dirac distributions,
444  * but with an efficient method to compute
445  * empirical cumulative distribution function.
446  */
447 class EmpiricalReal private[EmpiricalReal](points: Array[Double])
448 extends Distribution[Double] {
449     private val rnd = new Random
450     private val n = points.size
451     def sample = points(rnd.nextInt(n))
452     def integral(f: Double => Double) = {
453         var i = 0
454         var sum = 0.0
455         while (i < n) {
456             sum += f(points(i))
457             i += 1
458         }
459         sum / n
460     }
461     def cdf(t: Double): Double = {
462         var bs = java.util.Arrays.binarySearch(points, t)
463         if (bs >= 0) {
464             while (bs < (n - 1) && points(bs + 1) == t) bs += 1
465             bs += 1
466         } else {
467             bs = - bs - 1
468         }
469         bs.toDouble / n
470     }
471 }
472
473 object EmpiricalReal {
474     def apply(points: Iterable[Double]): EmpiricalReal = {
475         new EmpiricalReal(points.toArray.sorted)
476     }
477 }
478 }
```

```

479
480
481 /* #####
482 [!]      Stochastic processes and Markov chains
483 ##### */
484
485 /** Time discrete random process */
486 trait StochasticProcess[X] extends Distribution[Stream[X]] { outer =>
487
488   /** This process, stopped as soon as some predicate is fulfilled */
489   def stopped(hittingTimePredicate: X => Boolean): StochasticProcess[X] = {
490     new StochasticProcess[X] {
491       private def sampleHelper(s: Stream[X]): Stream[X] = {
492         val head #:: tail = s
493         if (hittingTimePredicate(head)) {
494           head #:: Stream.continually(head)
495         } else {
496           head #:: sampleHelper(tail)
497         }
498       }
499       def sample: Stream[X] = sampleHelper(outer.sample)
500       def integral(f: Stream[X] => Double) = ??? // easy, maybe later
501     }
502   }
503
504   /** pointwise mapping */
505   def mapPointwise[Y](f: X => Y): StochasticProcess[Y] =
506     new StochasticProcess[Y] {
507       def sample = (for (path <- outer) yield path.map(f)).sample
508       def integral(g: Stream[Y] => Double) = {
509         outer.integral(path => g(path.map(f)))
510       }
511     }
512
513   /** pointwise Markov kernel application */
514   def flatMapPointwise[Y](f: X => Distribution[Y]): StochasticProcess[Y] =
515     new StochasticProcess[Y] {
516       def sample = (for (path <- outer) yield path.map(x => f(x).sample)).sample
517       def integral(g: Stream[Y] => Double) = ??? // possible, but not needed now
518     }
519
520   /** pointwise zipping with other process */
521   def zipPointwise[Y](other: StochasticProcess[Y]): StochasticProcess[(X,Y)] = {
522     new StochasticProcess[(X,Y)] {
523       def sample =
524         (for (a <- outer; b <- other) yield (a zip b)).sample
525       def integral(g: Stream[(X,Y)] => Double) = ??? // possible, not needed now
526     }
527   }
528 }
529
530 /**
531  * Time discrete 'X'-valued Markov chain.
532  *
533  */
534 trait MarkovChain[X] extends StochasticProcess[X] { outer =>
535   /** Returns the initial distribution */
536   def initial: Distribution[X]
537   def next(current: X): Distribution[X]
538
539   /** Starts a new Markov chain at 'x' */
540   def startAt(x: X): StochasticProcess[X] = new StochasticProcess[X] {

```

B. Source code

```
541 // private val law = for {      // It looks correct, but it's not...
542 //   y <- outer.next(x)
543 //   tail <- outer.startAt(y)
544 // } yield x #:: tail
545
546 private def sampleTail(head: X): Stream[X] = {
547   val tailStart = outer.next(head).sample
548   tailStart #:: sampleTail(tailStart)
549 }
550
551 def sample = x #:: sampleTail(x)
552
553 def integral(f: Stream[X] => Double) = ??? // This seems rather difficult?
554 }
555
556 private val combinedLaw = {
557   val blah = initial
558   for (i <- initial; path <- startAt(i)) yield path
559 }
560
561 /**
562  * Starts a Markov chain with first valued chosen according to
563  * the initial distribution
564  */
565 def sample = combinedLaw.sample
566 def integral(f: Stream[X] => Double) = combinedLaw.integral(f)
567 }
568
569 /** A deterministic function reinterpreted as stochastic process */
570 abstract class DeterministicFunction[X] extends StochasticProcess[X] {
571   def apply(t: Int): X
572   def sample = Stream.from(0).map(t => this(t))
573   def integral(f: Stream[X] => Double) = f(sample)
574 }
575
576 /**
577  * Time-discrete random walk that is reflected
578  * at the bounds `min` and `max`.
579  */
580 class BoundedRandomWalk(min: Double, max: Double, jump: Double) extends {
581   val initial = RealUniform(min, max)
582 } with MarkovChain[Double] {
583   require(jump < (max - min))
584   def next(current: Double) = {
585     if (current + jump >= max) Dirac(current - jump)
586     else if (current - jump <= min) Dirac(current + jump)
587     else GenBernoulli(current + jump, current - jump)
588   }
589 }
590
591 /* #####
592    [!] Statistics
593    ##### */
594
595 /**
596  * A statistic of type `X,Y` is anything that can consume samples of type `X`
597  * and process them on the fly, yielding values of type `Y` in the end.
598  *
599  * For example, a structure that can consume lot of real numbers, and
600  * return their average in the end, is a statistic.
601  * A statistic should not occupy too much memory, if possible.
602  */
```

```

603 trait Statistic[-X, +Y] { outer =>
604   def consume(x: X): Unit
605   def result: Y
606   def prepend[Z](f: Z => X): Statistic[Z, Y] = new Statistic[Z, Y] {
607     def consume(z: Z) = outer.consume(f(z))
608     def result = outer.result
609   }
610   def map[Z](f: Y => Z): Statistic[X, Z] = new Statistic[X, Z] {
611     def consume(x: X) = outer.consume(x)
612     def result = f(outer.result)
613   }
614 }
615
616 class RealAverage extends Statistic[Double, Double] {
617   private var sum: Double = 0.0
618   private var number: Long = 0L
619   def consume(x: Double) = { sum += x; number += 1 }
620   def result = sum / number
621 }
622
623 class EcdfStatistic extends Statistic[Double, EmpiricalReal] {
624   private var allValues: List[Double] = Nil
625   def consume(x: Double) = { allValues ::= x }
626   def result = EmpiricalReal(allValues.toArray)
627 }
628
629 /* #####
630    [!] Partitions
631    ##### */
632
633 import scala.collection.immutable.Set
634
635 /** Extensional representation of a partition */
636 case class Partition[X](sets: Set[Set[X]]) {
637   override def toString = {
638     sets.map{
639       _.toList.map{_.toString}.sorted.mkString("{", ", ", ""}
640     }.toList.sorted.mkString("{", ", ", ""}
641   }
642
643   def totalSet = sets.flatten
644 }
645
646 object Partition {
647   /** Transforms an intensional representation of a partition into
648    * an extensional representation
649    * (This is essentially the function  $\mathcal{E}$ )
650    */
651   def groupBy[X, Y](what: Iterable[X], byWhat: X => Y): Partition[X] = {
652     val sets = what.toSet.groupBy(byWhat).values.toSet
653     Partition(sets)
654   }
655
656   def coarsest[X](total: Set[X]): Partition[X] = Partition(Set(total))
657   def finest[X](total: Set[X]): Partition[X] =
658     Partition(total.map{ x => Set(x) })
659 }
660
661
662
663
664

```

B. Source code

```
665
666
667 /* #####
668 [!] Random populations
669 ##### */
670
671 /* The goal of this chunk of code is to model (potentially) infinite streams
672 * of populations, without specifying any parentship relationships between
673 * different generations.
674 */
675
676 // A population is described by
677 // - the number of families,
678 // - an array of single-byte 'FamilyDescriptor's,
679 // - a 'FamilyStructure', that knows how to interpret the 'FamilyDescriptors'
680 type FamilyDescriptor = Byte
681
682 // The complete information about a random coalescent consists of a sequence
683 // of arrays with integer-triples as entries. Each triple contains the
684 // following information:
685 // - family index
686 // - index of individual within family
687 // - index of chromosome within individual
688 type FamilyIdx = Short
689 type IndividualIdx = Byte
690 type ChromosomeIdx = Byte
691
692 /**
693 * A 'FamilyStructure' describes possible types of families in a population.
694 * In some models (for example, monogamous diploid model), there will
695 * be just one type of family. However, for example for "alien bees",
696 * there will be multiple types of families, depending on the number of
697 * haploid males: '(1 queen, 1 male)', '(1 queen, 2 males)', ...,
698 * '(1 queen, 255 males)'.
699 */
700 trait FamilyStructure {
701   def numParents(descriptor: FamilyDescriptor): Int
702   def maxNumParents: Int
703   def randomDescriptor: Distribution[FamilyDescriptor]
704   def familyToString(descriptor: FamilyDescriptor): String
705   def fullCoordToString(f: FamilyIdx, i: IndividualIdx, c: ChromosomeIdx) =
706     "(f=%d,i=%d,c=%d)".format(f, i, c)
707
708   /** Suppose that we know that the parent family of an individual with
709   * index 'i' (internal index within family structure) is of type 'parent'.
710   * What are the possible ways for the individual 'i' to inherit its
711   * chromosomes from its parents?
712   *
713   * For example, in the monogamous diploid model with one male and one female
714   * as parents, there are four possible, equally probable assignments of the
715   * inherited chromosomes. If we mark father's chromosomes by '(a,b)' and
716   * mother's chromosomes by '(c,d)', then possible outcomes are:
717   * '(a,c)', '(a,d)', '(b,c)' and '(b,d)'.
718   */
719   def chromosomeInheritance(
720     i: IndividualIdx,
721     parent: FamilyDescriptor
722   ): Distribution[ChromosomeInheritance]
723
724   /** Supposing that a lineage is tracked far enough into the past,
725   * and it ends up in a family with the specified 'descriptor'.
726   * Which individual and which chromosome will the lineage hit with
```

```

727     * what probability?
728     *
729     * For example, if there is one father and one mother, both diploid,
730     * then each chromosome will be hit with probability '1/4'.
731     * On the other hand, if we have one diploid queen and 'D' haploid drones,
732     * then each chromosome of the queen will be hit with probability '1/3',
733     * while each drone will be hit with probability '1/3D'.
734     */
735     def equilibriumLineagePosition(
736         descriptor: FamilyDescriptor
737     ): Distribution[(IndividualIdx, ChromosomeIdx)]
738 }
739
740 /**
741  * For all our models, a family has essentially just one property:
742  * a natural number of "parents" (for example, number of drones + 1 queen for
743  * the bees/wasps). Therefore, a population is described by the number of
744  * families, and a single integer for each family (we shall call such an
745  * integer a "family descriptor").
746  * It's reasonable to assume that there aren't too many "family types" in each
747  * model, we restrict it to 256 in order to keep the representation compact.
748  */
749 case class Population(
750     familyStructure: FamilyStructure,
751     familyDescriptors: Array[FamilyDescriptor]
752 ) {
753     def numFamilies = familyDescriptors.size
754     lazy val numIndividuals = familyDescriptors.map{
755         d => familyStructure.numParents(d)
756     }.sum
757     override def toString = familyDescriptors.map{
758         d => familyStructure.familyToString(d)
759     }.mkString("Population[", ",", ",")
760     def apply(f: FamilyIdx) = familyDescriptors(f)
761 }
762
763 /**
764  * Generates an infinite stream of populations.
765  * Each population consists of a bunch of families, determined by their
766  * descriptors.
767  * The number of families is determined by the process 'numberOfFamilies'.
768  */
769 def randomPopulationHistory(
770     numberOfFamilies: StochasticProcess[Int],
771     familyStructure: FamilyStructure
772 ): StochasticProcess[Population] = numberOfFamilies.flatMapPointwise{
773     n => familyStructure.randomDescriptor.pow(n).map{
774         v => Population(familyStructure, v.toArray)
775     }
776 }
777
778 /* #####
779     [!] Random pedigrees
780     ##### */
781
782 /*
783  * Now we build random pedigrees on top of random population histories,
784  * by specifying parentship relations between adjacent generations.
785  */
786
787 /** A 'ParentFamilyChoice' is a data structure which, for each given
788     * individual '(f,i)' (individual from family 'f', with individual index 'i'),

```

B. Source code

```
789  * stores an index of a parent family from previous generation.
790  */
791  class ParentFamilyChoice (
792    val childPopulation: Population,
793    val parentPopulation: Population
794  ) extends ((FamilyIdx, IndividualIdx) => FamilyIdx) {
795
796    private val numFamilies = childPopulation.numFamilies
797    private val maxNumParents = parentPopulation.familyStructure.maxNumParents
798
799    private val parentFamily: Array[FamilyIdx] = {
800      new Array[FamilyIdx](numFamilies * maxNumParents)
801    }
802
803    def update(f: FamilyIdx, i: IndividualIdx, pf: FamilyIdx): Unit = {
804      parentFamily(f * maxNumParents + i) = pf
805    }
806
807    def apply(f: FamilyIdx, i: IndividualIdx): FamilyIdx =
808      parentFamily(f * maxNumParents + i)
809
810    override def toString = {
811      parentFamily.grouped(maxNumParents).map{
812        _.mkString(",")
813      }.mkString("PFC(", "|", ",")")
814    }
815  }
816
817  /** A 'OffspringNumberDistributionFactory' takes two inputs:
818   * total number of individuals in the current generation, and
819   * number of families in the previous generation.
820   * It returns a distribution of an 'Array[Int]' valued random variable, such
821   * that the size of the array corresponds to the number of families, the
822   * sum of entries of the array is equal to the total number of individuals,
823   * and furthermore, the entries of the array are exchangeable,
824   * natural-number-valued random variables.
825   */
826  trait OffspringNumberDistributionFactory {
827    def apply(
828      currentNumIndividuals: Int,
829      previousNumFamilies: Int
830    ): Distribution[Array[Int]]
831
832    /** This is essentially '\Phi_1(2)' */
833    def sameFamilyChoiceProbability(
834      currentNumIndividuals: Int,
835      previousNumFamilies: Int
836    ): Double
837  }
838
839  object WrightFisherFactory
840  extends OffspringNumberDistributionFactory {
841    /**
842     * Builds a special case of multinomial distribution, with all outcomes
843     * having the same probability.
844     */
845    def apply(currentNumIndividuals: Int, previousNumFamilies: Int) = {
846      new Distribution[Array[Int]] {
847        val rnd = new Random
848        def sample = {
849          val res = new Array[Int](previousNumFamilies)
850          for (i <- 0 until currentNumIndividuals) {
```

```

851         res(rnd.nextInt(previousNumFamilies)) += 1
852     }
853     res
854 }
855 def integral(f: Array[Int] => Double) = ??? // irrelevant...
856 }
857 }
858 /** `Phi 1(2)` */
859 def sameFamilyChoiceProbability(
860     currentNumIndividuals: Int,
861     previousNumFamilies: Int
862 ) = 1.0 / previousNumFamilies
863 }
864
865 /**
866  * A random pedigree is a `ParentFamilyChoice`-valued stochastic process,
867  * that is, it tells for each individual in each family in each generation
868  * what parent-family to choose.
869  */
870 def randomPedigree(
871     generations: Stream[Population],
872     offspringNumberFactory: OffspringNumberDistributionFactory
873 ): StochasticProcess[ParentFamilyChoice] =
874     new StochasticProcess[ParentFamilyChoice] {
875         def sample = {
876             val currentGenerations = generations
877             val parentGenerations = currentGenerations.tail
878             val adjacentGenerations = currentGenerations zip parentGenerations
879             for ((curr, prev) <- adjacentGenerations) yield {
880                 val offspringNumbers =
881                     offspringNumberFactory(curr.numIndividuals, prev.numFamilies).sample
882                 val sigma = UniformPermutation(curr.numIndividuals).sample
883                 val q = (for (
884                     (famIdx, numOff) <- (0 until prev.numFamilies) zip offspringNumbers;
885                     x <- Array.fill[FamilyIdx](numOff)(famIdx.toShort)
886                 ) yield x).toArray
887                 assert(q.forall{x => x >= 0},
888                     "prev.numFamilies = " + prev.numFamilies + "\n" +
889                     "q = " + q.mkString(",")
890                 )
891                 val qSigma = sigma.shuffle(q)
892                 assert(qSigma.forall{x => x >= 0})
893                 var r = 0
894                 val pfc = new ParentFamilyChoice(curr, prev)
895                 for (f <- (0 until curr.numFamilies).map(_.toShort).toArray) yield {
896                     val numPrts = curr.familyStructure.numParents(curr.familyDescriptors(f))
897                     for (i <- (0 until numPrts).map(_.toByte).toArray) yield {
898                         val parentFamilyIdx = qSigma(r)
899                         r += 1
900                         pfc(f, i) = parentFamilyIdx
901                     }
902                 }
903                 pfc
904             }
905         }
906     }
907 def integral(f: Stream[ParentFamilyChoice] => Double) = ??? // impractical
908 }
909
910 /**
911  * Population structure defines an an increasing process
912  * that corresponds to the internal time of the Kingman's coalescent.
913  */

```

B. Source code

```
913 * The following process defines time increments.
914 */
915 def virtualTimeIncrements(
916   generations: Stream[Population],
917   offspringNumberFactory: OffspringNumberDistributionFactory,
918   familyStructure: FamilyStructure
919 ): Stream[Double] = {
920   // This is 'c_N' divided by '\Phi_1(2)': we don't have to compute it
921   // manually, the Giry-monad does this job for us.
922   val averageSameChromosomeChoiceProb =
923     (for {
924       descr <- familyStructure.randomDescriptor
925       firstLineage <- familyStructure.equilibriumLineagePosition(descr)
926       secondLineage <- familyStructure.equilibriumLineagePosition(descr)
927     } yield (firstLineage == secondLineage)).prob{ b => b }
928
929   for ((curr, prev) <- generations zip generations.tail) yield {
930     val phil2 = offspringNumberFactory.sameFamilyChoiceProbability(
931       curr.numIndividuals,
932       prev.numFamilies
933     )
934     phil2 * averageSameChromosomeChoiceProb // This is our c_N
935   }
936 }
937
938 /**
939  * Cumulated sums of time increments
940  */
941 def virtualTime(
942   generations: Stream[Population],
943   offspringNumberFactory: OffspringNumberDistributionFactory,
944   familyStructure: FamilyStructure
945 ): Stream[Double] = {
946   val deltas = virtualTimeIncrements(
947     generations,
948     offspringNumberFactory,
949     familyStructure
950   )
951   deltas.scanLeft(0d){ case (prevSum, entry) => prevSum + entry }
952 }
953
954 /* #####
955    [!] Coalescents in random pedigrees
956    ##### */
957
958 /* Now we can simulate random coalescents in random pedigrees.
959  * We need a way to represent the outcomes of the
960  * Mendelian randomness experiments.
961  * This is what 'ChromosomeInheritance' is for.
962  */
963
964 /**
965  * A 'ChromosomeInheritance' is a function that determines how the genome of
966  * an individual is composed from the genome of individual's parents.
967  *
968  * It takes the index of a chromosome of the individual as input, and
969  * returns index of the parent, as well as index of a chromosome within the
970  * parent, that is copied by the individual.
971  *
972  * Example: Suppose we have a diploid individual (with chromosomes numbered
973  * 0 and 1)
974  * Suppose its parent family consists of a diploid mother (individual index 0)
```

```

975 * and a diploid father (with individual index 1).
976 * Then
977 * {{{
978 *   f(0) = (0,1)
979 *   f(1) = (1,0)
980 * }}}
981 * would be a valid `ChromosomeInheritance` function. It would tell us, that
982 * the first chromosome of the individual is the same as the second chromosome
983 * of the mother, and the second chromosome is the same as the first chromosome
984 * of the father.
985 */
986 trait ChromosomeInheritance
987 extends (ChromosomeIdx => (IndividualIdx, ChromosomeIdx))
988
989 /**
990 * Special `ChromosomeInheritance` for haploid individuals.
991 * Since there is just one chromosome, its index can be ignored.
992 */
993 case class ConstInheritance(i: IndividualIdx, c: ChromosomeIdx)
994 extends ChromosomeInheritance {
995   def apply(ignored: ChromosomeIdx) = (i, c)
996 }
997
998 /**
999 * Completely describes predecessors of a sample.
1000 * Corresponds to values of `X^{N,n}_g` in the proof.
1001 */
1002 case class FullState(
1003   state: Array[(FamilyIdx, IndividualIdx, ChromosomeIdx)],
1004   familyStructure: Option[FamilyStructure] = None // not strictly necessary
1005 ) {
1006   override def toString = if (familyStructure.isEmpty) {
1007     state.mkString("Full[","",""]")
1008   } else {
1009     state.map{x => familyStructure.get.fullCoordToString(x._1, x._2, x._3)}.
1010     mkString("Full[",";",""]")
1011   }
1012   def toPartition: Partition[Int] = Partition.groupBy(
1013     0 until state.size, idx => state(idx)
1014   )
1015   def apply(i: Int) = state(i)
1016   def sampleSize = state.size
1017 }
1018
1019 // Corresponds to the process `X^{N,n}_g` in the proof.
1020 def fullCoalescentHistory(
1021   sampleSize: Int,
1022   pedigree: Stream[ParentFamilyChoice]
1023 ): StochasticProcess[FullState] = new StochasticProcess[FullState] {
1024
1025   private def mendelianSampling(
1026     relevantIndividualCoords: Set[(FamilyIdx, IndividualIdx)],
1027     pfc: ParentFamilyChoice
1028   ): Map[(FamilyIdx, IndividualIdx), ChromosomeInheritance] = {
1029     (for ((f, i) <- relevantIndividualCoords) yield {
1030       // what is the parent family of the individual `(f,i)`?
1031       val predFamIdx = pfc(f, i)
1032       // get the descriptor of the parent family from `ParentFamilyChoice`
1033       val predFamDescr = pfc.parentPopulation(predFamIdx)
1034       // use the `FamilyStructure` to obtain the law of Mendelian
1035       // inheritance for this individual and this family type
1036       val familyStructure = pfc.parentPopulation.familyStructure

```

B. Source code

```

1037     val mendelianLaw =
1038         familyStructure.chromosomeInheritance(i, predFamDescr)
1039     // sample an assignment of chromosomes to parents and their
1040     // chromosomes
1041     val chromosomeInheritance = mendelianLaw.sample
1042     ((f, i), chromosomeInheritance)
1043   }).toMap
1044 }
1045
1046 private def sampleHelper(
1047     startingAt: FullState,
1048     remainingPedigree: Stream[ParentFamilyChoice]
1049 ): Stream[FullState] = {
1050     remainingPedigree.scanLeft(startingAt){ (s, pfc) =>
1051         val relevantIndividualCoords: Set[(FamilyIdx, IndividualIdx)] =
1052             s.state.map{ x => (x._1, x._2) }.toSet
1053         val relevantMendelianOutcomes =
1054             mendelianSampling(relevantIndividualCoords, pfc)
1055         val newFullState = Array.tabulate(s.sampleSize){ i =>
1056             // what chromosome does 'i' th marker point to?
1057             val (famIdx, indIdx, chrIdx) = s(i)
1058             // what is the parent family of the individual `(famIdx, indIdx)`?
1059             val predFamIdx = pfc(famIdx, indIdx)
1060             // what is the relevant outcome of the Mendelian experiment?
1061             val chromosomeInheritance = relevantMendelianOutcomes((famIdx, indIdx))
1062             // use the chromosomeInheritance to obtain parent index and index of
1063             // the chromosome within parent
1064             val (predIndIdx, predChrIdx) = chromosomeInheritance(chrIdx)
1065             // combine family index with parent index and chromosome index into
1066             // a new, completely unambiguous, coordinate of the 'i' th marker
1067             (predFamIdx, predIndIdx, predChrIdx)
1068         }
1069         FullState(newFullState, Some(pfc.parentPopulation.familyStructure))
1070     }
1071 }
1072
1073 def sample = {
1074     // start with a uniform injection
1075     val firstNumFamilies = pedigree(0).childPopulation.numFamilies
1076     val law_x0 =
1077     for (j <- UniformInjection(sampleSize, firstNumFamilies)) yield {
1078         FullState(
1079             Array.tabulate(sampleSize){i => (j(i).toShort, 0.toByte, 0.toByte)},
1080             Some(pedigree(0).parentPopulation.familyStructure)
1081         )
1082     }
1083     val realization_x0 = law_x0.sample
1084
1085     // use the sample helper to continue the stream
1086     sampleHelper(realization_x0, pedigree)
1087 }
1088
1089 def integral(f: Stream[FullState] => Double) = ??? // impractical
1090 }
1091
1092 // Corresponds to  $\frac{X}{N, n}_g$  in the proof
1093 def partitionCoalescentHistory(
1094     sampleSize: Int,
1095     pedigree: Stream[ParentFamilyChoice]
1096 ): StochasticProcess[Partition[Int]] =
1097     fullCoalescentHistory(sampleSize, pedigree).mapPointwise(_.toPartition)
1098 }

```

```

1099
1100
1101
1102 /* #####
1103 [!]          States and holding times representation
1104 #####*/
1105
1106 /**
1107  * State and holding time representation of a coalescent.
1108  * The lists 'states' and 'holdingTimes' store only the relevant entries
1109  * 'S_2,S_3,...,S_n' and 'H_2,H_3,...,H_n'.
1110  */
1111 class StatesHoldingTimes(
1112   val sampleSize: Int,
1113   val states: List[Partition[Int]],
1114   val holdingTimes: List[Double]
1115 ) {
1116   def mrcaTime = holdingTimes.sum
1117   override def toString = {
1118     (for ((h,s) <- (holdingTimes zip states).reverse) yield {
1119       "%2.3f %s".format(h,s)
1120     }).mkString("StatesTimes\n ", "\n ", "\n|") +
1121     " mrcaTime = " + holdingTimes.sum + "]"
1122   }
1123 }
1124
1125 object StatesHoldingTimes {
1126   /**
1127    * Builds a states-and-holding-times representation
1128    * from a stream of partitions and the virtual time.
1129    */
1130   def apply(
1131     sampleSize: Int,
1132     partitionHistory: Stream[Partition[Int]],
1133     virtualTime: Stream[Double]
1134   ): StatesHoldingTimes = {
1135     var lastSize = sampleSize + 1
1136     var lastJumpTime = -42.0
1137     var lastState = Partition.finest((0 to sampleSize).toSet)
1138     var states: List[Partition[Int]] = Nil
1139     var holdingTimes: List[Double] = Nil
1140     for ((s, t) <- partitionHistory zip virtualTime) {
1141       if (lastSize > s.sets.size) {
1142         // jump detected
1143         while (lastSize > s.sets.size) {
1144           holdingTimes ::= (t - lastJumpTime)
1145           lastJumpTime = t
1146           states ::= s
1147           lastSize -= 1
1148         }
1149         if (s.sets.size == 1) {
1150           return new StatesHoldingTimes(
1151             sampleSize,
1152             states.tail,
1153             holdingTimes.take(sampleSize - 1)
1154           )
1155         }
1156       }
1157     }
1158     throw new RuntimeException("Unexpectedly reached end of infinite stream.")
1159   }
1160 }

```

B. Source code

```
1161
1162 val Venus = '\u2640' // female
1163 val Mars = '\u2642' // male
1164 val Mercury = '\u263F' // hermaphrodite
1165
1166 /* #####
1167 [!] Meme model
1168 #####*/
1169
1170 object MemeFamilyStructure extends FamilyStructure {
1171   def numParents(ignore: Byte) = 2
1172   def maxNumParents = 2
1173   def randomDescriptor = Dirac(0.toByte) // there is only one type of family
1174   def familyToString(ignore: Byte) = "" + Venus + Mars
1175   def chromosomeInheritance(i: IndividualIdx, parentFamilyDescriptor: Byte):
1176     Distribution[ChromosomeInheritance] = {
1177       // structure of parent family is always the same, 'i' is also irrelevant:
1178       // we always just copy the meme either from mother, or from father.
1179       // Since both mother and father are "meme-haploid", the "chromosome"-index
1180       // is always 0.
1181       GenBernoulli(
1182         ConstInheritance(0,0), // inherit 0-th meme from mother
1183         ConstInheritance(1,0) // or 0-th meme from father
1184       )
1185     }
1186   def equilibriumLineagePosition(d: FamilyDescriptor):
1187     Distribution[(IndividualIdx, ChromosomeIdx)] =
1188       for (i <- GenBernoulli(0, 1)) yield (i.toByte, 0.toByte)
1189
1190   override def fullCoordToString(
1191     f: FamilyIdx,
1192     i: IndividualIdx,
1193     c: ChromosomeIdx
1194   ) = "(%d,%s)".format(f, (if (i == 0) (" " + Venus) else (" " + Mars)))
1195 }
1196
1197 /* #####
1198 [!] 'Duke' model (diploid, single locus, one male, one female)
1199 #####*/
1200
1201 case class DiploidInheritance(
1202   motherChromosome: ChromosomeIdx,
1203   fatherChromosome: ChromosomeIdx
1204 ) extends ChromosomeInheritance {
1205   def apply(ci: ChromosomeIdx) =
1206     if (ci == 0) (0.toByte, motherChromosome)
1207     else (1.toByte, fatherChromosome)
1208 }
1209
1210 object DukeFamilyStructure extends FamilyStructure {
1211   def numParents(ignore: FamilyDescriptor) = 2
1212   def maxNumParents = 2
1213   def randomDescriptor = Dirac(0.toByte)
1214   def familyToString(ignore: FamilyDescriptor) = "" + Venus + Mars
1215   def chromosomeInheritance(i: IndividualIdx, parentFamilyDescriptor: Byte):
1216     Distribution[ChromosomeInheritance] = {
1217       // without restriction of generality, the first gene is always
1218       // inherited from mother, the second from father
1219       FiniteUniform(Array(
1220         DiploidInheritance(0.toByte, 0.toByte),
1221         DiploidInheritance(0.toByte, 1.toByte),
1222         DiploidInheritance(1.toByte, 0.toByte),
```

```

1223     DiploidInheritance(1.toByte, 1.toByte)
1224   ))
1225 }
1226 def equilibriumLineagePosition(ignored: FamilyDescriptor):
1227   Distribution[(IndividualIdx, ChromosomeIdx)] =
1228     for {
1229       i <- GenBernoulli(0.toByte, 1.toByte)
1230       c <- GenBernoulli(0.toByte, 1.toByte)
1231     } yield (i, c)
1232
1233 override def fullCoordToString(
1234   f: FamilyIdx,
1235   i: IndividualIdx,
1236   c: ChromosomeIdx
1237 ) = "%d,%s,%s".format(f,
1238   (if (i == 0.toByte) (" " + Venus) else (" " + Mars)),
1239   c.toInt
1240 )
1241 }
1242
1243 /* #####
1244    [!] Polygynous fish model
1245    ##### */
1246
1247 case class FishInheritance(
1248   fatherChromosome: ChromosomeIdx,
1249   motherIdx: IndividualIdx,
1250   motherChromosome: ChromosomeIdx
1251 ) extends ChromosomeInheritance {
1252   def apply(ci: ChromosomeIdx) =
1253     if (ci == 0) (0.toByte, fatherChromosome)
1254     else (motherIdx, motherChromosome)
1255 }
1256
1257 /**
1258  * Family consisting of a single diploid father-fish
1259  * and a uniformly chosen number of 'minFemales' to 'maxFemales'
1260  * diploid females.
1261  *
1262  * Father-fish has index 0.
1263  * Females are numbered 1 to 'maxFemales'.
1264  * Family descriptor 'd' corresponds to a family with 'd' females.
1265  * The descriptor 'd=0' should never occur.
1266  */
1267 case class FishFamilyStructure(minFemales: Byte, maxFemales: Byte)
1268 extends FamilyStructure {
1269   require(minFemales > 0,
1270     "A fish family needs at least one female, but minFemales = " + minFemales)
1271   require(maxFemales >= minFemales,
1272     "Inconsistency: minFemales = " + minFemales +
1273     " maxFemales = " + maxFemales)
1274   def numParents(d: FamilyDescriptor) = (d.toInt + 1)
1275   def maxNumParents = maxFemales.toInt + 1
1276   def randomDescriptor = IntUniform(minFemales, maxFemales + 1).map{_.toByte}
1277   def familyToString(d: FamilyDescriptor) = Mars + (" " + Venus) * d.toInt
1278   def chromosomeInheritance(i: IndividualIdx, d: FamilyDescriptor):
1279     Distribution[ChromosomeInheritance] = {
1280       // without restriction of generality, the first gene is
1281       // inherited from the father-fish, the other gene is
1282       // inherited from the uniformly chosen mother-fish.
1283       for {
1284         fc <- GenBernoulli(0.toByte, 1.toByte)

```

B. Source code

```
1285     m <- IntUniform(0, d).map{ _ + 1 }
1286     mc <- GenBernoulli(0.toByte, 1.toByte)
1287   } yield FishInheritance(fc, m.toByte, mc)
1288 }
1289
1290 private def equilibriumHelper(d: FamilyDescriptor)(lineageInFather: Boolean):
1291   Distribution[(IndividualIdx, ChromosomeIdx)] = {
1292     if (lineageInFather) {
1293       GenBernoulli((0.toByte, 0.toByte), (0.toByte, 1.toByte))
1294     } else {
1295       for {
1296         m <- IntUniform(0, d).map{ _ + 1 }
1297         res <- GenBernoulli(0,1).map{ x => (m.toByte, x.toByte) }
1298       } yield res
1299     }
1300   }
1301
1302 private val EquilibriumLineageInMaleProb = 0.5
1303 def equilibriumLineagePosition(d: FamilyDescriptor):
1304   Distribution[(IndividualIdx, ChromosomeIdx)] =
1305     Bernoulli(EquilibriumLineageInMaleProb).flatMap{
1306       l => equilibriumHelper(d)(l)
1307     }
1308
1309 override def fullCoordToString(
1310   f: FamilyIdx,
1311   i: IndividualIdx,
1312   c: ChromosomeIdx
1313 ) = "%d,%s,%s".format(f,
1314   (if (i == 0.toByte) (" " + Mars) else (" " + Venus + i)),
1315   c.toInt
1316 )
1317 }
1318
1319 /* #####
1320    [!] Alien-ants model
1321    ##### */
1322
1323 // Ants have two different inheritance mechanisms for queen and drones.
1324 // Since drones are haploid, we can reuse 'ConstInheritance' defined above,
1325 // but the queen needs yet another inheritance strategy.
1326
1327 /**
1328  * A queen inherits one chromosome from it's mother queen, and
1329  * one chromosome from a particularly lucky drone.
1330  * Since there is only one queen, we need only queen chromosome index.
1331  * Since every drone is haploid, we need only drone's individual index.
1332  */
1333 case class AntQueenInheritance(
1334   queenChromosomeIdx: ChromosomeIdx,
1335   luckyDroneIdx: IndividualIdx
1336 ) extends ChromosomeInheritance {
1337   def apply(ci: ChromosomeIdx) =
1338     if (ci == 0) (0.toByte, queenChromosomeIdx)
1339     else (luckyDroneIdx, 0.toByte)
1340 }
1341
1342 /**
1343  * Fertile individuals that contribute to the genome of a colony are:
1344  * - a single diploid female queen
1345  * - multiple haploid male drones
1346  * There can be between 'minDrones' and 'maxDrones' drones.
```

```

1347 *
1348 * Queen has individual index 0.
1349 * Drones are numbered with indices 1 to 'maxDrones' (inclusively).
1350 * Family descriptor 'd' corresponds to a colony with 'd' drones.
1351 * The descriptor 'd=0' should never occur.
1352 */
1353 case class AntsColonyStructure(minDrones: Byte, maxDrones: Byte)
1354 extends FamilyStructure {
1355   require(minDrones > 0,
1356     "An ant colony needs at least one drone, but minDrones = " + minDrones)
1357   require(maxDrones >= minDrones,
1358     "Inconsistency: minDrones = " + minDrones +
1359     " maxDrones = " + maxDrones)
1360   def numParents(d: FamilyDescriptor) = (d.toInt + 1)
1361   def maxNumParents = maxDrones.toInt + 1
1362   def randomDescriptor = IntUniform(minDrones, maxDrones + 1).map{_.toByte}
1363   def familyToString(d: FamilyDescriptor) = Venus + (" " + Mars) * d.toInt
1364   def chromosomeInheritance(i: IndividualIdx, d: FamilyDescriptor):
1365     Distribution[ChromosomeInheritance] = {
1366       // queen and drones are quite different beasts... treat them separately
1367       if (i == 0) {
1368         // queen
1369         for {
1370           qci <- GenBernoulli(0.toByte, 1.toByte)
1371           lucky <- IntUniform(0, d).map{ _ + 1 }
1372         } yield AntQueenInheritance(qci, lucky.toByte)
1373       } else {
1374         // all drones are kind-of half-clones of the queen
1375         for {
1376           qci <- GenBernoulli(0.toByte, 1.toByte)
1377         } yield ConstInheritance(0.toByte, qci)
1378       }
1379     }
1380
1381   private def equilibriumHelper(d: FamilyDescriptor)(lineageInQueen: Boolean):
1382     Distribution[(IndividualIdx, ChromosomeIdx)] = {
1383       if (lineageInQueen) {
1384         GenBernoulli((0.toByte, 0.toByte), (0.toByte, 1.toByte))
1385       } else {
1386         IntUniform(0, d).map{ i => ((i + 1).toByte, 0.toByte) }
1387       }
1388     }
1389
1390   private val EquilibriumLineageInQueenProb = 2.0 / 3.0
1391   def equilibriumLineagePosition(d: FamilyDescriptor):
1392     Distribution[(IndividualIdx, ChromosomeIdx)] =
1393       Bernoulli(EquilibriumLineageInQueenProb).flatMap{
1394         l => equilibriumHelper(d)(l)
1395       }
1396
1397   override def fullCoordToString(
1398     f: FamilyIdx,
1399     i: IndividualIdx,
1400     c: ChromosomeIdx
1401   ) = ("%d,%s,%s").format(f,
1402     (if (i == 0.toByte) (" " + Venus) else (" " + Mars + i)),
1403     c.toInt
1404   )
1405 }
1406
1407
1408

```

B. Source code

```
1409
1410
1411
1412 /* #####
1413 [!] Code formatting
1414 ##### */
1415
1416 /*
1417 * The code in this section makes some cosmetic changes on the code itself:
1418 * it skims through the file, finds all lines marked by a exclamation mark in
1419 * square brackets, and inserts a simple line-based index at the beginning of
1420 * the file.
1421 *
1422 * It has nothing to do with genetics or stochastic processes whatsoever.
1423 */
1424 import scala.io.StdIn.readLine
1425 val SectionTag = "]![".reverse
1426
1427 /** Reads source code from std-in, inserts an actualized index between
1428 * the INDEX-tags at the beginning of the file.
1429 */
1430 def createIndex(): Unit = {
1431     var line: String = ""
1432     var state = "beforeIndex"
1433     var beforeIndex: List[String] = Nil
1434     var afterIndex: List[String] = Nil
1435     var sections: List[(String,Int)] = Nil
1436     var lineNumber = 1
1437     while ({line = readLine(); line != null}) {
1438         state match {
1439             case "beforeIndex" => {
1440                 if(line.contains("[INDEX]")) {
1441                     state = "skippingIndex"
1442                 }
1443                 beforeIndex ::= line
1444                 lineNumber += 1
1445             }
1446             case "skippingIndex" => {
1447                 if (line.contains("/INDEX")) {
1448                     state = "normal"
1449                     afterIndex ::= line
1450                     lineNumber += 1
1451                 }
1452             }
1453             case "normal" => {
1454                 if (line.contains(SectionTag)) {
1455                     val title = line.drop(line.indexOf(SectionTag) + 3).trim
1456                     sections ::= (title, lineNumber)
1457                 }
1458                 afterIndex ::= line
1459                 lineNumber += 1
1460             }
1461         }
1462     }
1463     for (l <- beforeIndex.reverse) println(l)
1464     println()
1465     for ((title, lineNumber) <- sections.reverse) {
1466         val lineString = "" + (lineNumber + sections.size + 2)
1467         print(title)
1468         print("." * (80 - title.size - lineString.size))
1469         println(lineString)
1470     }
}
```

```

1471     println()
1472     for (l <- afterIndex.reverse) println(l)
1473 }
1474
1475 /* #####
1476 [!]           Parameter parsing
1477 ##### */
1478
1479 // Just parsing command line arguments,
1480 // nothing particularly interesting here...
1481
1482 class ArgsOption(
1483     val names: List[String],
1484     val help: String,
1485     val default: String,
1486     val isFlag: Boolean = false,
1487     val regex: String = "[_-,0-9a-zA-Z()]+")
1488 {
1489     var value: Option[String] = None
1490     def immediateAction(): Unit = {}
1491     def get: String = value.getOrElse(default)
1492     def set(a: String): Unit = { value = Some(a) }
1493     override def toString = names.mkString("/")
1494     def verboseDescription: String = {
1495         names.sortBy(_.size).last + " = " + get
1496     }
1497 }
1498
1499 class ArgsOptions(opts: List[ArgsOption]) {
1500     def parse(arguments: Array[String]): Unit = {
1501         var justParsed: Option[ArgsOption] = None
1502         for (a <- arguments) {
1503             if (!justParsed.isEmpty && !justParsed.get.isFlag) {
1504                 if (a.matches(justParsed.get.regex)) {
1505                     justParsed.get.set(a)
1506                     justParsed = None
1507                 } else {
1508                     println("Invalid argument for option '" + justParsed.get + "':")
1509                     println(">>>" + a + "<<<")
1510                     println("Expected regex: " + justParsed.get.regex)
1511                     System.exit(1)
1512                 }
1513             } else {
1514                 opts.find{ o => o.names.contains(a) } match {
1515                     case None => {
1516                         println("Unrecognized option >>>" + a + "<<<")
1517                         System.exit(1)
1518                     }
1519                     case Some(o) => {
1520                         justParsed = Some(o)
1521                         o.immediateAction()
1522                         if (o.isFlag) o.set("true")
1523                     }
1524                 }
1525             }
1526         }
1527     }
1528
1529     /**
1530     * Returns modification of this 'ArgsOptions' with one
1531     * additional, automatically generated help option.
1532     */

```

B. Source code

```
1533 def withHelp(  
1534   generalHelpIntro: String,  
1535   generalHelpOutro: String  
1536 ): ArgsOptions = {  
1537   val helpOption = new ArgsOption(  
1538     List("-h", "-?", "--help", "-help"),  
1539     "Prints this help and exits", "false",  
1540     true  
1541   ) {  
1542     override def immediateAction = {  
1543       println(generalHelpIntro)  
1544       for (o <- opts) {  
1545         println(" " + o.names.mkString(" / "))  
1546         val indented = (  
1547           for (l <- o.help.split("\n")) yield ("    " + l)  
1548         ).mkString("\n")  
1549         println(indented)  
1550       }  
1551       println(generalHelpOutro)  
1552       System.exit(0)  
1553     }  
1554   }  
1555   new ArgsOptions(helpOption :: opts)  
1556 }  
1557  
1558 def apply(optName: String): String = {  
1559   opts.find{_.names.contains(optName)} match {  
1560     case Some(hit) => hit.get  
1561     case None => {  
1562       println("Could not find value for command line option " + optName)  
1563       System.exit(0)  
1564       throw new Exception  
1565     }  
1566   }  
1567 }  
1568  
1569 def verboseDescription: String = {  
1570   (for (o <- opts) yield {  
1571     o.verboseDescription  
1572   }).mkString("\n")  
1573 }  
1574 }  
1575  
1576 val createIndexOption = new ArgsOption(  
1577   List("--create-index"),  
1578   "Reads source code from STDIN, outputs formatted source code with added " +  
1579   "index to STDOUT.", "false", true  
1580 ) {  
1581   override def immediateAction(): Unit = {  
1582     createIndex()  
1583     System.exit(0)  
1584   }  
1585 }  
1586  
1587 val cli = new ArgsOptions(List(  
1588   new ArgsOption(List("--pedigrees", "-p"),  
1589     "Number of generated pedigrees.\nDefault: '-p 10'", "10",  
1590     false, "[1-9][0-9]*"),  
1591   new ArgsOption(List("--coalescents", "-c"),  
1592     "Number of sampled coalescents.\nDefault: '-c 256'", "256",  
1593     false, "[1-9][0-9]*"),  
1594   new ArgsOption(List("--sample-size", "-n"),
```

```

1595     "Sample size.\nDefault: '-n 2'", "2",
1596     false, "[1-9][0-9]*"),
1597 new ArgOption(List("--num-families","-N"),
1598     "Number of families per generation.\nDefault: '-N 100'", "100",
1599     false, "[1-9][0-9]*"),
1600 new ArgOption(List("--num-families-variation"),
1601     "Relative variation of number of families.\nDefault: 0\n" +
1602     "Examples: '--num-families 1000 --num-families-variation 0.5' will \n" +
1603     "produce a pedigree where the number of families per generation varies\n" +
1604     "between 500 and 1500. Accepts only numbers from [0,1).",
1605     "0.0", false, "0\\. [0-9]+"),
1606 ),
1607 new ArgOption(List("--model","-m"),
1608     "Family model. Available options are:\n" +
1609     "  Meme\n" +
1610     "  Duke\n" +
1611     "  Fish(<minFemales>,<maxFemales>)\n" +
1612     "  Ants(<minDrones>,<maxDrones>)\nDefault: '-m Meme'\n" +
1613     "Examples: '-m Duke', '-m Fish(7,15)', '-m Ants(10,20)'",
1614     "Meme", false,
1615     ""(Meme)|(Duke)|(Fish\\([0-9]+,[0-9]+\\))|(Ants\\([0-9]+,[0-9]+\\))""),
1616 ),
1617 new ArgOption(List("--exp-1-cdf"),
1618     "Outputs values of distribution function of Exp_1 in the first column.",
1619     "false", true),
1620 ),
1621 new ArgOption(List("--mrca-ecdf"),
1622     "Output values of empirical cumulative distribution \n" +
1623     "function of the MRCA time. One column per pedigree is produced. ",
1624     "false", true),
1625 ),
1626 new ArgOption(List("--mrca-avg"),
1627     "Output average MRCA time (one for each pedigree)", "false", true),
1628 ),
1629 new ArgOption(List("--verbose","-v"),
1630     "Generates verbose output.", "false", true),
1631 new ArgOption(List("--show-environment"),
1632     "Dumps first 'g' populations and parent family choices.\n" +
1633     "Works only in verbose mode.\n" +
1634     "It's preferable to set '-p 1' on multicore machines, otherwise the \n" +
1635     "output for different pedigrees can get scrambled.\n" +
1636     "Don't use it with large 'N'. \n" +
1637     "Default: '--show-environment 0'\n" +
1638     "Example: '--show-environment 20' shows first 20 generations",
1639     "0", false, "[1-9][0-9]*"),
1640 new ArgOption(List("--track-progress"),
1641     "Prints progress information to STDERR.\n" +
1642     "Looks really cool with multi-core CPU's.", "false", true),
1643 new ArgOption(List("--comment"),
1644     "Prepends the specified prefix to each line of verbose output.\n" +
1645     "Try '--comment \"#\"' for gnuplot or '--comment \"%\"' for LaTeX",
1646     "%", false, ".+"),
1647 ),
1648 new ArgOption(List("--plot-resolution"),
1649     "Step width for ECDF plots. Default: '--plot-resolution 0.01'",
1650     "0.01",
1651     false,
1652     "[0-9]+(\\. [0-9]+)?"),
1653 ),
1654 new ArgOption(List("--plot-max"),
1655     "Step width for ECDF plots. Default: '--plot-max 3.0'",
1656     "3.0",

```

B. Source code

```
1657     false,
1658     "[0-9]+(\\.[0-9]+)?"
1659   ),
1660   new ArgsOption(List("--only-populations"),
1661     "Don't simulate any coalescents. Just generate the populations, \n" +
1662     "output intrinsic time and number of families (two columns).",
1663     "false",
1664     true
1665   ),
1666   new ArgsOption(List("--only-coalescence-probabilities"),
1667     "Don't simulate any coalescents and ignores all other settings. \n" +
1668     "Just print the coalescence \n" +
1669     "probabilities conditioned on the event that two lineages hit \n" +
1670     "the same family for all available models.",
1671     "false",
1672     true
1673   ),
1674   createIndexOption
1675 ).withHelp(
1676   "Simulates gene genealogies in fixed pedigrees.\n Available options are:",
1677   "\nA typical invocation might look as follows: \n\n" +
1678   "  scala coalescentSimulation.scala \\n\n" +
1679   "  --sample-size 2 --num-families 100 --num-families-variation 0.75 \\n\n" +
1680   "  --model 'Ants(5,10)' --pedigrees 20 --coalescents 1000 \\n\n" +
1681   "  --verbose --comment '#' \\n\n" +
1682   "  --exp-1-cdf --mrca-ecdf --track-progress\n\n" +
1683   "These settings describe a model with family structure of an ant colony\n" +
1684   "where a single queen and 5 to 10 males contribute to the genome of each\n" +
1685   "colony. \n" +
1686   "  The number of colonies in each generation varies between 25 and 175. \n" +
1687   "This command would generate 20 different pedigrees, and simulate 1000\n" +
1688   "coalescents on each pedigree. \n" +
1689   "Each coalescent would start with 2 active lineages.\n" +
1690   "  The program would output all the settings, prefixed by an '#'-symbol.\n" +
1691   "Then it would print a big table, with t-values in the first column, \n" +
1692   "CDF of Exp_1 in the second column, and then 20 further columns with \n" +
1693   "ECDF's of pair coalescence times (one column per pedigree).")
1694 )
1695
1696 /* #####
1697  [!]      Entry point, running the experiment
1698  ##### */
1699
1700 val augmentedArgs = if (args.isEmpty) Array("--help") else args
1701 cli.parse(augmentedArgs)
1702
1703 val verboseMode = cli("--verbose").toBoolean
1704 val trackProgress = cli("--track-progress").toBoolean
1705 val showEnvironment = cli("--show-environment").toInt
1706 val commentPrefix = cli("--comment")
1707 def printVerbose(s: String): Unit = if (verboseMode) {
1708   println(s.split("\n").map{l => commentPrefix + " " + l}.mkString("\n"))
1709 }
1710 def printProgress(s: String): Unit = if (trackProgress) {
1711   System.err.println(s)
1712 }
1713
1714 // dump the settings if necessary
1715 printVerbose(cli.verboseDescription)
1716
1717 val numFamilies = cli("--num-families").toInt
1718 val variation = cli("--num-families-variation").toDouble
```

```

1719 if (variation < 0.0 || variation >= 1.0) {
1720     println("Invalid --num-families-variation: " + variation +
1721         " (expected 0 <= x < 1)")
1722     System.exit(1)
1723 }
1724 val plotMax = cli("--plot-max").toDouble
1725
1726 val numFamiliesProcess = if (variation == 0.0) {
1727     printVerbose("Number of families is constant " + numFamilies)
1728     new DeterministicFunction[Int] { def apply(t: Int) = numFamilies }
1729 } else {
1730     val minFamilies = (numFamilies * (1 - variation)).toInt
1731     val maxFamilies = (numFamilies * (1 + variation)).toInt
1732     // 2.0 is to keep it crashing into walls frequently,
1733     // square root is to keep the relative variance roughly the same at all
1734     // time scales.
1735     val jumpSize = 2.0 * math.sqrt(numFamilies)
1736     printVerbose("Number of families is a bounded random walk \n" +
1737         "with values between " + minFamilies + " and " + maxFamilies + "\n" +
1738         "making jumps of size " + jumpSize)
1739     (new BoundedRandomWalk(minFamilies, maxFamilies, jumpSize)).mapPointwise{
1740         _.toInt
1741     }
1742 }
1743
1744 val familyStructure = {
1745     val model = cli("--model").trim
1746     if (model == "Meme") {
1747         printVerbose("Family structure: 'Meme', all families look the same.")
1748         MemeFamilyStructure
1749     } else if (model == "Duke") {
1750         printVerbose("Family structure: 'Duke', all families look the same.")
1751         DukeFamilyStructure
1752     } else if (model.startsWith("Fish") || model.startsWith("Ants")) {
1753         val modelName = model.take(4)
1754         val intParams = model.drop(5).dropRight(1).split(",").map(_.toInt)
1755         printVerbose(
1756             "Chosen model: '" + modelName +
1757             "' with parameters: " + intParams.mkString(" ")
1758         )
1759         if (intParams.size != 2) {
1760             println("Expected 2 integer params, but got " + intParams.size)
1761             System.exit(1)
1762         }
1763         if (!intParams.forall{p => p >= 0 && p < 127}) {
1764             println("Invalid family model params: expected values between 0 and 126")
1765             System.exit(1)
1766         }
1767         val minOpp = intParams(0).toByte
1768         val maxOpp = intParams(1).toByte
1769         if (modelName == "Fish") {
1770             FishFamilyStructure(minOpp, maxOpp)
1771         } else if (modelName == "Ants") {
1772             AntsColonyStructure(minOpp, maxOpp)
1773         } else {
1774             throw new Exception(
1775                 "Unrecognized parameterized model name: " + modelName
1776             )
1777         }
1778     } else {
1779         throw new Exception("Unrecognized model: " + model)
1780     }
1781 }

```

B. Source code

```
1781
1782 val generationsProcess = randomPopulationHistory(
1783     numFamiliesProcess,
1784     familyStructure
1785 )
1786
1787 val numPedigrees = cli("--pedigrees").toInt
1788 val numCoalescents = cli("--coalescents").toInt
1789 val sampleSize = cli("--sample-size").toInt
1790
1791 val statMrcaEcdf = cli("--mrca-ecdf").toBoolean
1792 val statMrcaAvg = cli("--mrca-avg").toBoolean
1793
1794 // run experiment only if it's really required...
1795 val simulateCoalescents = statMrcaEcdf || statMrcaAvg
1796 val simulateOnlyPopulations = cli("--only-populations").toBoolean
1797 val showOnlyCoalescenceProbs = cli("--only-coalescence-probabilities").toBoolean
1798
1799 if (simulateOnlyPopulations && simulateCoalescents) {
1800     println("No coalescents can be simulated when option --only-populations " +
1801         "is active. Please remove --mrca-ecdf, --mrca-avg and all other flags " +
1802         "that require simulation of coalescents."
1803     )
1804     System.exit(2)
1805 }
1806
1807 if (showOnlyCoalescenceProbs && simulateCoalescents) {
1808     println("No coalescents can be simulated when " +
1809         "option --only-coalescence-probabilities " +
1810         "is active. Please remove --mrca-ecdf, --mrca-avg and all other flags " +
1811         "that require simulation of coalescents."
1812     )
1813     System.exit(3)
1814 }
1815
1816 // This is the main experiment: simulation of coalescents in fixed pedigrees
1817 if (simulateCoalescents) {
1818
1819     val experimentStartTime = System.currentTimeMillis
1820     val pedigreeProgress = new Array[Double](numPedigrees)
1821     var lastProgressDisplay = experimentStartTime
1822     def showPedigreeProgress(force: Boolean = false): Unit = {
1823         if (trackProgress) {
1824             val now = System.currentTimeMillis
1825             if (now - lastProgressDisplay > 250 || force) {
1826                 lastProgressDisplay = now
1827                 printProgress("Progress after " + (now - experimentStartTime) + " ms :")
1828                 for (pIdx <- 0 until numPedigrees) {
1829                     val percentageFloat = pedigreeProgress(pIdx) * 100
1830                     val percentage = percentageFloat.toInt
1831                     printProgress(
1832                         "%4d ".format(pIdx) + ("#" * percentage) +
1833                         (" " * (100 - percentage)) + " " +
1834                         "%6.2f %%".format(percentageFloat)
1835                     )
1836                 }
1837             }
1838         }
1839     }
1840
1841     // each pedigree can be treated completely independently -> parallelize
1842     val statsForAllPedigrees = for (pIdx <- (0 until numPedigrees).par) yield {
```

```

1843     var labeledStats: List[(String,Statistic[StatesHoldingTimes, _])] = Nil
1844     if (statMrcaEcdf) {
1845         labeledStats ::=
1846             ("--mrca-ecdf", (new EcdfStatistic()).prepend{ tree => tree.mrcaTime })
1847     }
1848     if (statMrcaAvg) {
1849         labeledStats ::=
1850             ("--mrca-avg", (new RealAverage()).prepend{ tree => tree.mrcaTime })
1851     }
1852
1853     val fixedGenerations = generationsProcess.sample
1854
1855     val intrinsicTime = virtualTime(
1856         fixedGenerations,
1857         WrightFisherFactory,
1858         familyStructure
1859     )
1860
1861     val fixedPedigree =
1862         randomPedigree(fixedGenerations, WrightFisherFactory).sample
1863
1864     if (showEnvironment > 0) {
1865         printVerbose("Random environment " + pIdx)
1866         printVerbose("Generations: ")
1867         for (exampleGen <- fixedGenerations.take(showEnvironment))
1868             printVerbose(exampleGen.toString)
1869         printVerbose("Pedigree: ")
1870         for (examplePfc <- fixedPedigree.take(showEnvironment))
1871             printVerbose(examplePfc.toString)
1872     }
1873
1874     val coalescentFullLaw = partitionCoalescentHistory(
1875         sampleSize,
1876         fixedPedigree
1877     )
1878     val coalescentLaw = for (path <- coalescentFullLaw) yield {
1879         StatesHoldingTimes(sampleSize, path, intrinsicTime)
1880     }
1881     for (cIdx <- 0 until numCoalescents) {
1882         val coalescentRealization = coalescentLaw.sample
1883         for ((_,s) <- labeledStats) {
1884             s.consume(coalescentRealization)
1885         }
1886         pedigreeProgress(pIdx) = (cIdx + 1) / numCoalescents.toDouble
1887         if (cIdx % 10 == 0) showPedigreeProgress()
1888     }
1889     labeledStats
1890 }
1891 showPedigreeProgress(true)
1892 val experimentEndTime = System.currentTimeMillis
1893 val experimentTime = (experimentEndTime - experimentStartTime) / 1000.0
1894
1895 printVerbose("Total time = %10.2f sec = %10.2f min".format(
1896     experimentTime, experimentTime / 60.0))
1897
1898 // output results of the statistics
1899 val plotResolution = cli("--plot-resolution").toDouble
1900 if (plotResolution <= 0.0) {
1901     println("Non-positive plot resolution: " + plotResolution)
1902     System.exit(1)
1903 }
1904

```

B. Source code

```
1905 val explcdf = cli("--exp-1-cdf").toBoolean
1906 def selectStats[Y](label: String): List[Statistic[StatesHoldingTimes,Y]] = {
1907   (for {
1908     labeledStats <- statsForAllPedigrees
1909     (statLabel, stat) <- labeledStats
1910     if (statLabel == label)
1911   } yield stat.asInstanceOf[Statistic[StatesHoldingTimes, Y]]).toList
1912 }
1913
1914 if (statMrcaAvg) {
1915   printVerbose("Results --mrca-avg:")
1916   for (s <- selectStats("--mrca-avg")) {
1917     println(s.result)
1918   }
1919 }
1920
1921 if (statMrcaEcdf) {
1922   printVerbose("Results --mrca-ecdf:")
1923   val ecdfs = selectStats[EmpiricalReal]("--mrca-ecdf").map{_.result}
1924   val numSteps = (plotMax / plotResolution).toInt
1925   for (k <- (0 to numSteps)) {
1926     val t = k * plotResolution
1927     printf("%2.6f ", t)
1928     if (explcdf) {
1929       printf("%2.6f ", 1 - math.exp(-t))
1930     }
1931     for (ecdf <- ecdfs) {
1932       printf("%2.6f ", ecdf.cdf(t))
1933     }
1934     println()
1935   }
1936 }
1937 }
1938
1939 // Simulating only populations: printing
1940 // a column with virtual time, and a column with varying
1941 // number of families `(N_g)_g`.
1942 if (simulateOnlyPopulations) {
1943   printVerbose("Results --only-populations " +
1944     "(intrinsic time, number of families):"
1945   )
1946   val fixedGenerations = generationsProcess.sample
1947   val intrinsicTime = virtualTime(
1948     fixedGenerations,
1949     WrightFisherFactory,
1950     familyStructure
1951   )
1952   for ((t,g) <- intrinsicTime zip fixedGenerations) {
1953     if (t > plotMax) {
1954       System.exit(0) // enough, just quit
1955     } else {
1956       printf("%2.6f %2.6f\n".format(t, g.numFamilies.toDouble / numFamilies))
1957     }
1958   }
1959 }
1960
1961 /* #####
1962    [!]          Sanity checks for theoretical formulas
1963    ##### */
1964
1965 if (showOnlyCoalescenceProbs) {
1966   println("Ants")
1967 }
```

```

1967 for (maxDrones <- 1 to 10) {
1968   for (minDrones <- 1 to 5) {
1969     if (maxDrones < minDrones) {
1970       printf("(-----,-----) ")
1971     } else {
1972       val fs = AntsColonyStructure(minDrones.toByte, maxDrones.toByte)
1973       val theoreticalValue =
1974         (2 + (minDrones to maxDrones).map{
1975           x => 1.0/x
1976         }.sum / (maxDrones - minDrones + 1)) / 9
1977       val automaticValue =
1978         (for {
1979           descr <- fs.randomDescriptor
1980           firstLineage <- fs.equilibriumLineagePosition(descr)
1981           secondLineage <- fs.equilibriumLineagePosition(descr)
1982         } yield (firstLineage == secondLineage)).prob{ b => b }
1983       printf("(%5.4f,%5.4f) ", theoreticalValue, automaticValue)
1984     }
1985   }
1986   println()
1987 }
1988 println("Fish")
1989 for (b <- 1 to 10) {
1990   for (a <- 1 to 5) {
1991     if (b < a) {
1992       printf("(-----,-----) ")
1993     } else {
1994       val fs = FishFamilyStructure(a.toByte, b.toByte)
1995       val theoreticalValue =
1996         (1 + (a to b).map{
1997           x => 1.0/x
1998         }.sum / (b - a + 1)) / 8
1999       val automaticValue =
2000         (for {
2001           descr <- fs.randomDescriptor
2002           firstLineage <- fs.equilibriumLineagePosition(descr)
2003           secondLineage <- fs.equilibriumLineagePosition(descr)
2004         } yield (firstLineage == secondLineage)).prob{ b => b }
2005       printf("(%5.4f,%5.4f) ", theoreticalValue, automaticValue)
2006     }
2007   }
2008   println()
2009 }
2010
2011 }

```


Bibliography

- [1] Patrick Billingsley. *Convergence of Probability Measures*. Wiley Interscience, second edition edition, 1999.
- [2] Richard Durrett. *Probability Models for DNA Sequence Evolution*. Probability and its applications. Springer, second edition edition, 2008.
- [3] Michèle Giry. A categorical approach to probability theory. In B. Banaschewski, editor, *Categorical Aspects of Topology and Analysis*, volume 915 of *Lecture Notes in Mathematics*, pages 68–85. Springer Berlin Heidelberg, 1982.
- [4] Achim Klenke. *Wahrscheinlichkeitstheorie*. Springer, second edition edition, 2008.
- [5] Zhi Yong Liu, Zi Long Wang, Wei Yu Yan, Xiao Bo Wu, Zhi Jiang Zeng, and Zachary Y. Huang. The sex determination gene shows no founder effect in the giant honey bee, *apis dorsata*. *PLoS ONE*, 7(4):e34436, 04 2012.
- [6] M. Möhle. A convergence theorem for markov chains arising in population genetics and the coalescent with selfing. *Adv. Appl. Prob.*, 30:493–512, 1998.
- [7] Martin Möhle. Stochastische populationsgenetik, Wintersemester 2007/2008.
- [8] Martin Möhle and Serik Sagitov. A classification of coalescent processes for haploid exchangeable population models. *Ann. Probab.*, 29(4):1547–1562, 10 2001.
- [9] Martin Möhle and Serik Sagitov. Coalescent patterns in diploid exchangeable population models. *Journal of Mathematical Biology*, 47(4):337–352, 2003.
- [10] M. R. Morris O. Rios-Cardenas. volume VIII of *Tropical Biology and Conservation Management*. Encyclopedia of Life Support Systems (EOLSS).
- [11] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala*. Artima Press, 2008.
- [12] John Wakeley, Leandra King, Bobbi S. Low, and Sohini Ramachandran. Gene genealogies within a fixed pedigree, and the robustness of Kingman’s coalescent. *Genetics*, 190:1433–1445, April 2012.