University of Tübingen
Faculty of Science
Department of Computer Science

Master Thesis Bioinformatics

Simulating and estimating the effect of gene transfer on bacterial pangenomes

Jules Kreuer

01.06.2024

Reviewers

Dr. Franz Baumdicker
(Bioinformatics)
Institute for Bioinformatics and Medical Informatics
University of Tübingen

Prof. Dr. Daniel Huson
(Bioinformatics)
Institute for Bioinformatics and Medical Informatics
University of Tübingen

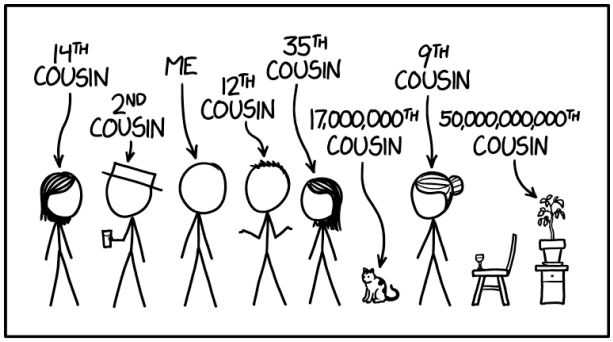
Kreuer, Jules:
Simulating and estimating the effect of gene transfer on bacterial pangenomes.
Master Thesis Bioinformatics
University of Tübingen
Thesis period: 01.12.2023 - 01.06.2024

Abstract

Horizontal gene transfer (HGT) plays a significant role in shaping the genetic landscape of bacterial populations. In contrast to the more common vertical gene transfer, horizontal gene transfer allows the lateral exchange of genes. To study the impact of HGT on bacterial gene frequency spectra, we have extended existing mutation models within the open-source software *msprime* by incorporating a gene gain and loss model using the Infinitely Many Genes model approach. The ancestry and mutation simulation is then extended to support HGT events. Additionally, the model is adjusted to fix its otherwise random ancestry simulation to specified trees, which is essential for parameter estimation and fitting the simulation to real data. We then develop an innovative simulation-based testing framework to determine whether a gene frequency spectrum results from neutral evolution. Finally, this framework is validated, and real-world parameters are estimated using pangenome data. The final model, documentation, simulation data and examples are available on GitHub: github.com/not-a-feature/pangenome-gene-transfer-simulation

Acknowledgements

[print version only]



REALLY, *EVERY* GATHERING IS A FAMILY REUNION.

xkcd.com/2608/

Contents

List of Figures

List of Tables

List of Abbreviations

Chapter 1: Introduction

The genetic profile of a population does not only just describe its characteristics, it also provides insights into its hereditary and evolutionary history. In eukaryotes, including humans, the evolutionary histories of genes are largely intertwined due to recombination and sexual reproduction. This contrasts with prokaryotic populations, where genes often follow a dominant clonal lineage [LN21]. However, Horizontal Gene Transfer (HGT) can disrupt this pattern by allowing genes to move between lineages and individuals, introducing genetic variation [Sch23].

The Gene Frequency Spectrum (GFS), which describes the number of genes observed in k individuals within a population, reflects these evolutionary effects. Under conditions of clonal inheritance, the GFS responds to the underlying tree structure, in which genes are passed on to descendants and thus accumulate within a subtree [BP14]. Deviations between the observed GFS and the GFS expected under a neutral clonal model may indicate evolutionary processes beyond simple mutations, indicating possible HGT. Understanding the dynamics of HGT is usefull for several reasons. Firstly, it improves our understanding of microbial evolution, including the role lateral gene flow plays in achieving genetic diversity. Unlike the slow, gradual accumulation of mutations in a purely clonal model, HGT can introduce substantial genetic changes in a short time frame, accelerating adaptation and evolution. Secondly, horizontal gene transfer has profound implications for public health, particularly with regard to antibiotic resistance. Resistance-conferring genes can spread rapidly through bacterial populations via HGT, undermining treatment efforts and leading to the emergence of multi-drug resistant strains [Bur15].

Methodologically, integrating HGT into existing models of genetic evolution improves their accuracy and predictive power. Traditional models based on clonal inheritance often fail to capture the complexity of microbial evolution.

This thesis provides a comprehensive biological and mathematical background for all concepts used and extends the open source software *msprime* and *tskit* [Bau+21]. Specifically, the mutation model of *msprime* is extended by introducing a computational approach of the infinite gene model [BP14] through a gene gain and loss model. This model allows the simulation of neutral evolution. Using this model, a neutrality test is developed to determine whether the gene frequency spectrum results from neutral evolutionary processes. Furthermore, the ancestry and mutation simulation is extended by the introduction of horizontal gene transfer events and a tree fixation mechanism. The fixation allows the simulation of gene conversion, recombination and HGT while maintaining a given tree structure as a backbone. With that, the effect of gene conversion and HGT on the GFS is assessed. Finally, we estimate the strength of HGT by recording simulation parameters across different bacterial species which minimises the error between the simulated and real Gene Frequency Spectrum. This analysis uses pangenome data from the NCBI, allowing for a detailed comparison between simulated results and observed genetic patterns.

Chapter 2: Background

2.1 Biological Background

- **Genes**: A gene is a basic unit of heredity in a living organism. Genes, made up of Deoxyribonucleic acid (DNA), act as instructions for making molecules called proteins. Genes can be gained or lost from bacterial genomes by various mechanisms, affecting the characteristics and abilities of the organism.

- **Genome**: The genome of an organism is the complete set of DNA, including all its genes. In bacteria, the genome is usually a single circular DNA molecule called a chromosome, which contains all the genetic information. Bacteria can also contain smaller DNA molecules called plasmids, which often carry extra genes. Plasmids are also usually circular. The bacterial genome can vary greatly in size and gene content between different species.

- **Bacterial reproduction**: Bacteria reproduce primarily by asexual binary fission, in which a single cell replicates its organelles and genome before dividing into two identical daughter cells. This process allows rapid population growth and results genetic uniformity.

2.1.1 Genetic Diversity

Despite the clonal reproduction process in bacteria, genetic variation exists. This diversity stems inter alia from migration, evolutionary selection but also from mutations, gene conversion and horizontal gene transfer.

Mutations are spontaneous changes in the DNA sequence, such as single nucleotide polymorphisms, insertions, deletions, and duplications. These changes can result from errors in DNA replication, exposure to environmental factors such as radiation, or the activities of mobile genetic elements.

An important but less frequent mechanism in genetics is gene conversion. It involves the non-reciprocal transfer of genetic information between homologous sequences, which can result in the replacement of a section of one DNA strand with a sequence from a homologous strand. Gene conversion can occur within a chromosome (allelic conversion) or between different chromosomes (ectopic conversion).

During gene conversion, one segment of DNA is replaced by a highly similar (homologous) segment from a different position or DNA strand, meaning the two sections end up becoming identical after the process. Gene conversion often occurs during DNA replication or repair. There, single strands from two similar DNA sequences temporarily align. One strand in the pair breaks, and DNA polymerase starts copying the sequence of the other strand to repair the break. This replaces the corresponding section in the broken strand.

The length of the repaired segment varies and may span a single gene or several genes, depending on the size of the original break and the duration of the strand pairing. As the broken strand of DNA is repaired, the two DNA molecules separate. Unlike traditional 'crossing over' in genetic recombination, gene conversion is unidirectional, with one strand acting as the 'donor' and overwriting the sequence of the other 'recipient' strand.

Horizontal Gene Transfer (HGT), also known as lateral gene transfer, is the process of transferring genetic material between different organisms, bypassing the standard method of genetic inheritance from parent to offspring [SHG15] [Sch23]. The concept of HGT emerged in the late 1940s and initially focused on microorganisms, particularly bacteria and archaea. In certain cases, the effect was also observed in eukaryotes such as plants [KP08].

In bacterial HGT, there are several key mechanisms by which it may occur:

1. **Transformation**: This involves the uptake and incorporation of foreign DNA or RNA into a cell. It occurs naturally in bacteria, but can be induced artificially by exposing cells to heat or electrical pulses.

2. **Transduction**: In this process, bacterial DNA is transferred from one bacterium to another via a phage. Phages are simple entities consisting of genetic material enclosed in a protein shell (capsid) that enables them to attach to the host cell by specifically recognising host proteins. The transduction process commences when a phage infects a bacterial cell, turning it into a donor. The phage integrates its genome into the bacterial chromosome. During excision, part of the bacterial chromosome is accidentally cut out along with the phage DNA. This excised chromosomal DNA, now packaged with the phage DNA inside the phage capsule, leaves the donor cell. These new phages then infect other cells, transforming them into recipients and transferring the extrachromosomal DNA from the original donor. One of the best known phages in this context is the lambda phage, famous for its ability to infect E. coli [Sch23].

3. **Gene Transfer Agents**: These are virus-like structures found in certain bacteria that facilitate gene transfer. They act similarly to bacteriophages, but instead of carrying their own genetic material, GTAs package random fragments of their host's DNA.

4. **Bacterial Conjugation**: This process uses direct cell-to-cell contact to transfer DNA from a donor cell to a recipient cell. The most common gene transfer system relies on a piece of DNA called an F (fertility) plasmid. The donor cell, which carries the F plasmid, forms a pilus, a tube-like structure that extends to make contact with the recipient cell. Once contact is made, the F plasmid replicates in the donor cell. One of the plasmid strands is then transferred through the pilus into the recipient cell. If the plasmid integrates into the genome, it is likely that neighbouring genes will also be transferred. It is important to note that this process is usually unidirectional, i.e. genetic material can be transferred from the donor to the recipient, but not vice versa. After the transfer, the recipient cell synthesises a complementary

strand to the incoming single-stranded DNA, forming a complete plasmid. This genetic material can be incorporated into the recipient's genome or exist as an independent plasmid within the cell.

These events may lead to genetic changes that can, but do not always, result in different phenotypes. In some species, such as Buchnera aphidicola, genetic variation accumulates mainly through point mutations. This characteristic, together with its very small genome size of 500kb and only 350-590 genes [CPM19], makes Buchnera aphidicola of particular interest for the simulation part of this work.

In other species, such as methicillin-resistant Staphylococcus aureus (MRSA), the effect of HGT is crucial. This strain is thought to have acquired resistance genes horizontally from animal-associated bacteria. Once a strain acquires resistance, it multiplies and evolves as it becomes more likely to spread between patients and healthcare facilities. This phenomenon is not limited to specific bacterial lineages, but occurs across different strains and sometimes even Kingdoms [Ric+11] leading to the emergence of diverse populations. In contrast to species with a strong clonal or HGT effect, organisms such as Streptococcus pneumoniae and Neisseria meningitidis exhibit high rates of homologous recombination [Ton+23] [Bur15].

2.2 Simulation Background

The latter described simulation algorithm uses a concept called the pangenome. The pangenome is the set of all genes carried by all individuals in a population, reflecting the genetic variation and potential for adaptation. It can be divided into two main parts:

1. **Core Genome** G_c: The core genome consists of genes present in every individual of the population. These genes are often essential for survival and reproduction and known to be highly conserved.

2. **Dispensable / Individual Genome** G_i: These are genes that are not present in all individuals but may confer selective advantages in certain environments or conditions.

The concept of a pangenome was first proposed and analysed for pathogenic strains of Streptococcus agalactiae by Tettelin et al. in 2005 [Tet+05]. Their analysis showed that approximately 80% of the genes in a single S. agalactiae genome are shared by all individuals and are therefore part of G_c. However, each fully sequenced genome contains unique genes that are not found in other individuals. This aligns with the distributed genome hypothesis, proposed by Ehrlich et al. in 2005 [Ehr+10] stating that no single organism possesses the entire set of genes of its species. Species with a small dispensable genome are typically referred to as 'closed', while those with a highly variable dispensable genome are referred to as 'open'.

2.2.1 Wright-Fisher Model

The Wright-Fisher model is a key concept in population genetics and is used to describe the evolution of the genetic composition of a population over time [Mes16]. It is particularly suitable for prokaryotes, such as bacteria and archaea, as its concepts are centred on asexual reproduction.

The model assumes a constant population size and that the population evolves through discrete, non-overlapping generations. This means that each generation is distinct from the previous one. This size, denoted N_e, is the effective number of individuals able to contribute to the next generation rather than the total number of individuals. For sexually reproducing organisms, N_e is strongly influenced by location, physical distance, but also by mating and social aspects [LN21].

For example, the effective population size for humans is estimated to be 10,000, despite a census population size of 8.1 billion (8.1×10^9). Similarly, for Prochlorococcus, a marine cyanobacterium responsible for 10% of global oxygen production, N_e is set to a relatively small 1.68×10^7 [Che+21], although the actual population size is around 3×10^{27} [Flo+13].

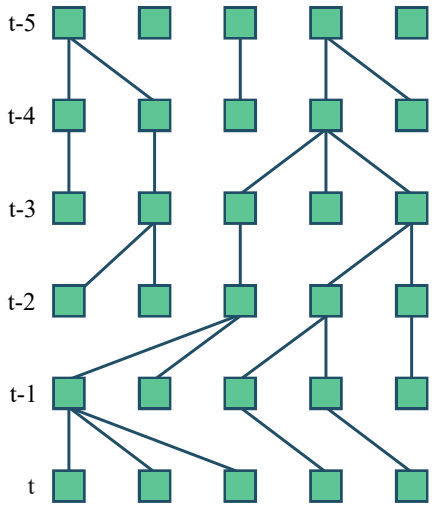


Figure 2.1: Wright-Fischer model with 5 generations. Every individual chooses one parent at random.

The Wright-Fisher model is best described backwards in time. Assuming neutral clonal (i.e. asexual) reproduction, with no selection, mutation or migration, each individual of generation t chooses a single parent from the previous generation $t-1$. This means that several individuals can have the same parent, and each parent has an equal chance of survival and reproduction, as seen in figure 2.1. If two individuals have a common ancestor, their lineages are said to coalesce.

The process of selecting parents in each generation follows a uniform random pattern. Therefore, the probability of a given individual being selected as a parent is $1/N$ and, conversely, the probability of that individual not being selected is $1-1/N$. Modelling this selection forwards in time, the number of offspring can be represented as a binomial distribution $X \sim B(n = N, p = 1/N)$. For large populations ($N >> 20$), the number of offspring an individual has follows a Poisson

distribution. This is due to the Poisson limit theorem, which allows the underlying binomial distribution to be approximated by the Poisson distribution for sufficiently large N:

$$P[X = k] = \lim_{n\to\infty} \binom{n}{k} p_n^k (1 - p_n)^{n-k} = e^{-\lambda} \frac{\lambda^k}{k!} \tag{2.1}$$

The average number an individual is chosen as a parent is $E[X] = n \cdot p = 1$. As the expected value of a Poisson distribution is:

$$E[X] = \sum_{k=0}^{\infty} k \cdot e^{-\lambda} \frac{\lambda^k}{k!} = \lambda \tag{2.2}$$

we know that under this model, the number of offspring of an individual has, is approximately Poisson distributed with $\lambda = 1$.

2.2.2 Kingman coalescent

The Kingman coalescent is a stochastic process that models the genealogical history of a sample of genes from a population and is closely related to the Wright-Fisher model. It is also known as the standard neutral coalescent model [LN21].

By examining all possible events of the (haploid) Wright-Fisher model in the immediate ancestry of a generation, Kingman found that a single coalescent event between two individuals is the most probable event. Averaging over multiple generations and considering all possible combination if individuals, the probability of a coalescent event is then given by

$$p_c = \binom{n}{2} \frac{\sigma^2}{N} + O\left(\frac{1}{N^2}\right) \tag{2.3}$$

where n is the number of samples, σ^2 is the variance of the number of offspring of a single individual [Kin82]. The binomial coefficient $\binom{n}{2}$ describes the number of possible lineage pairs. This equation assumes large populations where the exact probability is approximated by the first term; the term $O(1/N^2)$ represents the remaining coalescence of two or more lineages. The largest element of this power series expansion is proportional to $1/N^2$. As $N \to \infty$, the additional terms become negligible, leaving the first term alone.

As time is rescaled by the population size N, and as N approaches infinity, the time to Most Recent Common Ancestor (MRCA) of two samples converges to an exponential distribution $f(t) = e^{-t}$. The rate of coalescence between two lineages on the new timescale is thus 1, meaning that in a population of n individuals with $n(n-1)/2$ possible pairs, the rate of coalescence is $n(n-1)/2$.

If this model is simulated backwards in time and n lineages remain, the exponential waiting time is calculated with a rate of $n(n-1)/2$. This rate corresponds to the number of possible pairs of lineages that can coalesce, as each pair has a chance of coalescing into a single line. At the time of coalescence, two lines are randomly selected from the remaining n lines and merged into one. The simulation continues until only one line remains, the MRCA.

2.2.3 Gene Frequency Spectrum

The GFS is a method of summarising genetic diversity within a population by classifying genes according to their frequency among individuals. It is similar to the allele frequency spectrum. Taking the number of genes present in exactly k different individuals within a sample of size n gives the gene frequency class $G_k^{(n)}$. For instance, in a population of 100 individuals, the gene frequency class of $G_5^{(100)}$ represents the number of genes found in exactly 5 of those individuals.

For random trees without horizontal gene transfer the expected value $E[G_k]$ is given by

$$\mathbb{E}[G_k] = \frac{\theta}{k(n-1+\rho)(n-k+\rho)} \tag{2.4}$$

which gives us, depending on θ and ρ, a distinct L or U shape. As can be seen in figure 3.2, higher ρ cause gene loss events to dominate over gene gains, resulting in an L-shape with few high-frequency genes.

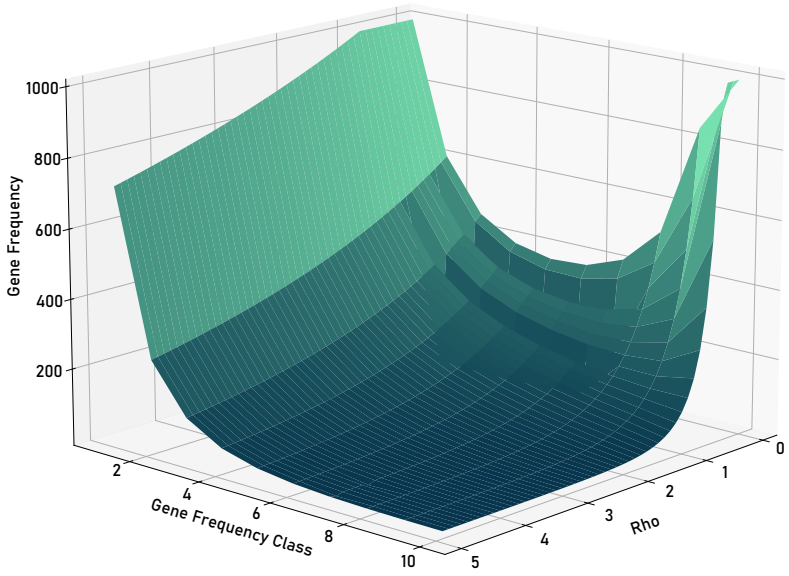


Figure 2.2: Expected Gene Frequency Spectrum for different gene loss rates ρ at fixed gene gain $\theta = 1000$ for 10 samples.

If, on the other hand HGT is present, Baumdicker et al. [BP14] estimated the expected GFS for large populations n and $\rho > 0, \theta > 0, \gamma \geq 0$ to be

$$\mathbb{E}\left[G_k^{(n)}\right] = \frac{\theta}{k} \frac{(n)_k \downarrow}{(n-1+\rho)_k \downarrow} \left(1 + \sum_{m=1}^{\infty} \frac{(k)_m \uparrow \gamma^m}{(n+\rho)_m \uparrow m!}\right) \tag{2.5}$$

with $(a)_{b\uparrow} := a(a+1)\cdots(a+b-1)$ and $(a)_{b\downarrow} := a(a-1)\cdots(a-b+1)$. In other words, genes appear with higher frequency, leading to a closed genome.

2.2.4 Infinitely Many Genes Model

The Infinitely Many Genes (IMG) model [BHP10] [BHP12] is the cornerstone of this work. It allows us to place gene gain and loss events along the ancestral lineages of individuals on the Kingman coalescent.

In this model, a gene can only be gained once, but has the potential to be lost multiple times independently on different branches. In addition, gene gain is exclusively associated with mutation rather than horizontal gene transfer.

To keep this model simple, the core genes G_c, which are essential for survival and reproduction, are ignored.

Beyond G_c, there exists an infinite pool of genes I with $G_c \cap I = \varnothing$. The genome of individual i in the sample, $1 \leq i \leq N$, contains the set of genes $G_i \subseteq I$ that are not necessary for the individuals to survive.

1. **Gene Gain** There is a probability μ before reproduction that an individual will acquire a new gene from the environment, which is assumed to have never been present in the population's gene pool before.

2. **Gene Loss** Each gene in the dispensable genome G_i of an individual i has a probability v of being lost before the reproduction event.

As we are working with the Kingman coalescent, time is measured in units of N_e instead of generations. Thus, the mutation rates μ and v are rescales in line with the genealogies so that $\theta = \lim_{N\to\infty} N \cdot \mu$ and $\rho = \lim_{N\to\infty} N \cdot v$

Following Baumdicker et al. [BHP10] we therefore get the average number of genes, in the dispensable genome, by the formula:

$$A = \frac{1}{n} \sum_{i=1}^{n} |G_i| \tag{2.6}$$

For A as defined above, the expected value is given by:

$$\mathbb{E}[A] = \frac{\theta}{\rho} \tag{2.7}$$

where θ and ρ are the gene gain and loss rates, respectively.

The IMG model differs from the traditional Finitely Many Genes (FMG) model, in which genes can be repeatedly gained and lost within evolutionary lineages. In the IMG framework, we assume an infinite pool of potential genes, whereas the FMG model, on the other hand, assumes a finite set of M possible gene types. Each gene gained is selected from this pool, so it is possible for the same gene to be gained several times during the evolutionary process. The overall rate at which genes are gained is $\theta = a \cdot M$, where a represents the gain rate of individual gene type. The IMG model can be seen as a limiting case of the FMG model, where the size of the gene pool M approaches infinity as the rate of gain of a particular gene a approaches zero, while the overall rate of gain θ remains constant.

Both models treat genes, gene gains and gene losses as independent entities / events, ignoring any positional effects or genomic order, such as the simultaneous

loss of entire genomic regions containing multiple genes. Furthermore, both models assume constant rates of gene gain and loss over time, despite known variations in these rates.

In this model, Horizontal Gene Transfer events between any two individuals i and j occur at a rate of γ/N for each gene $u \in G_i^N(t)$ in the genome of individual i at time t. During such an event, the gene u is duplicated from the donor i to the recipient j, resulting in the addition of u to $G_j^N(t)$, the gene set of j, while maintaining u in $G_i^N(t)$.

If the recipient j already has the gene u, the transfer event has no effect because the model excludes paralogous gene creation. While it is possible for bacteria to transfer multiple genes at once, this effect is neglected in this work as and each HGT event is responsible for only one gene.

2.2.5 msprime and tskit

msprime is a fast and flexible coalescent-based simulator. It can efficiently simulate the genealogies (ancestral recombination graphs) of individuals under different evolutionary models and is based on *tskit* which provides a compact data format for storing and manipulating tree sequences [KEM16] [Bau+21]. Both provide a python interface with C backbone. In the following work we will extend and use the *msprime* and *tskit* software packages.

To ensure clarity and precision, we provide a summary of their key terms and concepts:

- **Tree**: Refers to a genealogical tree outlining the relationships between a collection of genomes at a given chromosomal location.
- **Node**: A node is a branching point in a tree at a given time. At this node, several segments have come together. Non-branching nodes can exist.
- **Segment**: A segment represents a section of the genome. Each segment contains a left and right position indicating the boundaries within the genome, a reference node, and a pointer to the next and previous segments in the chain. Together, these segments form a chain that can (but are not required to) cover the entire genome.
- **Lineage**: If the head (left-most) segment of a segment chain is present in our population, this chain is considered a lineage.
- **Edge**: An edge is defined as a tuple marking a parent-child relationship between nodes across the boundaries (left, right) of a given segment.
- **Site**: A site is a unique position on the genome, associated with an allele.
- **Mutation**: This term refers to a change in state at a site that occurs above a particular node. Each mutation is linked to a specific site and node and has a defined derived state.

At the heart of *msprime* is the so-called 'hudson simulation loop', which iteratively models events that alter ancestral relationships. Initially, a separate lineage is created for each individual. Each lineage each consist of a segment spanning the entire genome and is associated with a unique (leaf) node. The head of each lineage is referenced in the population, a hashmap for quick access. The simulation loop continues as long as there are multiple lineages and the simulation time is less than a specified end time.

1. **Calculate event times**: As first step, the waiting time for each of the next possible events is calculated. These values range from 0 to positive infinity and follow an exponential distribution based on the input parameters. Gene gains are initialised with a rate of θ, losses with a rate of ρ, and gene conversion events with κ. It should be noted that ρ and κ are parameters that are relative to the total number of sites s.

2. **Determine next event**: The function identifies the next event based on the minimum of the calculated times (`min_time`).

3. **Execute event**:

   - Gene conversion ('GCI', 'GCL'): A small sub-segment (alpha) is cut from a randomly selected segment. This copied sub-segment is inserted into the population and forms a new, independent lineage.
   - Recombination ('REC'): A breakpoint of a random line is selected, and a segment is split at this position. By removing the link in the segment chain and adding the new orphaned chain to the population, this segment chain is treated as a separate lineage in the rest of the simulation.
   - Common Ancestor Event ('CA'): Two random lines are selected. A new node is inserted in the tree to represent this common ancestor, where the two lineages are merged. Edges are added that span the width of overlapping segments of both lineages. These edges will later be used for mutation simulation.

4. **Table building** After the simulation, the resulting tree is passed to the *tskit* library which uses a series of linked tables,

5. **Mutation simulation**: If desired, mutations can be placed along the edge using different models.

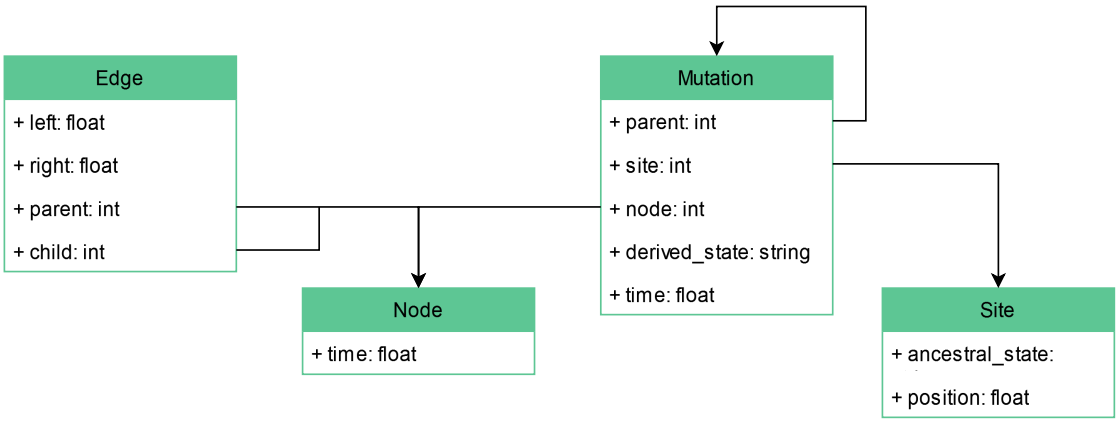


Figure 2.3: Relation of the tables managed by *tskit*.

The tables created during the simulation store the underlying data and are managed by *tskit*. Figure 2.3 shows the 4 main tables and omits some columns for simplicity. It is important to note that each row in a table has an indirect identifier based on its position, which can be references in other tables.

As nodes represent branching points in the tree where several segments have come together, the Node Table stores the time of these points. The Edge Table describes parent-child relationships between nodes over specific genomic intervals, with the left column denoting the inclusive start position and the right column denoting the exclusive end position. The Mutation table records mutation details within the genome:

- Parent: Identifier for the parent mutation.
- Site: Identifier of the mutation site.
- Node: Identifier for the node below which the mutation is observed.
- Derived State: The allelic state resulting from the mutation.
- Time: The time of occurrence.

The Site Table maps locations on the genome where allelic states are observed:

- Position: The exact genomic location. As we are using a discrete genome, the location must fall on an integer, although rational numbers are supported.
- Ancestral state: The allelic state at the root of the tree, before any mutations.

To store a complete tree structure, only the edge and Node Tables are required. If mutations are added, both an additional Mutation table and a Site Table must be added. The sum of all tables is a `TableCollection` to which the *tskit* interface provides indirect access via `RowObjects` or direct access to the underlying *numpy* arrays.

Chapter 3: Method and Implementation

3.1 Gene Model

This chapter introduces a function called `gene_model`, which is an extension of the *msprime* library (v. 1.3.1) [dev24]. The final model, documentation and examples are available on GitHub[1] This function simulates the evolutionary process of gene gain and loss by, among other things, adapting the `sim_mutations` function using a custom matrix mutation model.

In this adaptation, instead of simulating single nucleotide mutations, each event is either a complete gene gain or loss. Therefore, we use a Markov model with two distinct states for each site: 'absent' (0) and 'present' (1) with a possible transition between them. As *msprime* works with a finite genome with s sites, this approach ignores Dollo's law of irreversibility and contradicts the IMG, stating that a lost gene cannot be regained. To keep the probability of a gene being lost and regained low and follow the IMG, a large genome with many sites is used.

The model uses a matrix to determine the probability of transitions between these two states, influenced by the rates of gene gain θ and gene loss ρ. While gene gain is defined as a rate over the whole genome, gene loss only applies to individual sites. It must therefore be rescaled by the total number of sites s.

In addition, *msprime* operates with a predetermined event rate. Therefore, the probabilities of gene gain and loss are adjusted proportionally:

$$\begin{aligned} P_\theta &= \frac{\theta}{\theta + \rho \cdot s} \\ P_\rho &= \frac{\rho \cdot s}{\theta + \rho \cdot s} \end{aligned} \tag{3.1}$$

Since *msprime* only traces lineages up to the MRCA, this node needs a certain number of genes, otherwise the gene frequency spectrum will be skewed towards less frequent genes. This can be done by using a root distribution of present genes.

Using the Poisson limit theorem, it is possible to approximate binomial distributions for large samples and small probabilities by a Poisson distribution. This allows us to model the distribution of genes in the MRCA node as a Poisson distribution around the expected average number of genes:

$$n_{\text{present}} = \text{Poisson}(\frac{\theta}{\rho}) \tag{3.2}$$

The most efficient way to mark genes as 'present' is to modify the sites table in *tskit* directly. By marking only the present genes and omitting the absent ones,

[1] github.com/not-a-feature/pangenome-gene-transfer-simulation

further reduction in run time is achieved. Since the required number of present genes is already established at the root node, an absent state should be returned whenever *msprime* draws from the root distribution.

Performance analysis shows that adding all genes simultaneously using the build in `add_columns` method, focusing only on 'present' genes, is significantly faster. It outperforms the other approaches - setting both present and absent genes, or setting each position individually - by a factor of 2 to 10 in speed.

3.2 Horizontal Gene Transfer

Following the logic of the standard simulation loop, the model first calculates a random exponential waiting time weighted by the sum of segment length. If, among all other possible events, HGT is the one with the shortest waiting time, the model randomly selects an existing individual (line) and a recipient segment (`y`) along that individual (Step I in 3.1). Two breaks are placed at a random position within segment `y`, creating a new segment (`alpha`). `alpha` has a length of one and represents a single gene. This segment is now duplicated (`alpha'`) and modified so that it has no explicit origin, previous or subsequent segment associations, reflecting its status as a gene transfer line (Step II in 3.1). The donor segment is named `alpha'` or *alpha prime*. The number of segments at a given position are tracked and stored in a binary search tree (AVL tree). By duplicating `alpha`, this number needs to be increased by one. The key characteristic of an AVL tree is its height-balancing property: the heights of the two child subtrees of any node differ by at most one. This ensures that the tree remains balanced, maintaining a logarithmic height relative to the number of nodes, n. Consequently, operations such as lookup, insertion, and deletion can be performed in $O(\log n)$ time in both average and worst cases. Following these adjustments, the rate maps are updated with the new lineages. In certain edge-cases the handling of this process requires special attention.

If the alpha segment exactly matches the boundaries of the y segment (i.e., `y.left == alpha.left and y.right == alpha.right`), which can occur in horizontal gene transfer or gene conversion lineages, there is no need to insert breaks or change the recipient segment.

Furthermore, if the selected recipient segment is the first segment in the chain and its left breakpoint is equal to the left boundary of the recipient segment, then alpha must replace the recipient as the head of the lineage. Otherwise, an inconsistent segment chain would be created.

A new node (`HM`) of the newly created class `HORIZONTAL_MERGE` is added to act as the new parent of the original segments. Since a free-floating segment (`alpha'`) is not possible, it is linked to another new node (`HG`) of class `HORIZONTAL_GENE` (Step II), which has a direct parent (`HP`) of class `HORIZONTAL_PARENT`. This newly formed ancestor is then treated in the same way as any other line within the model, including possible future mergers with other lines.

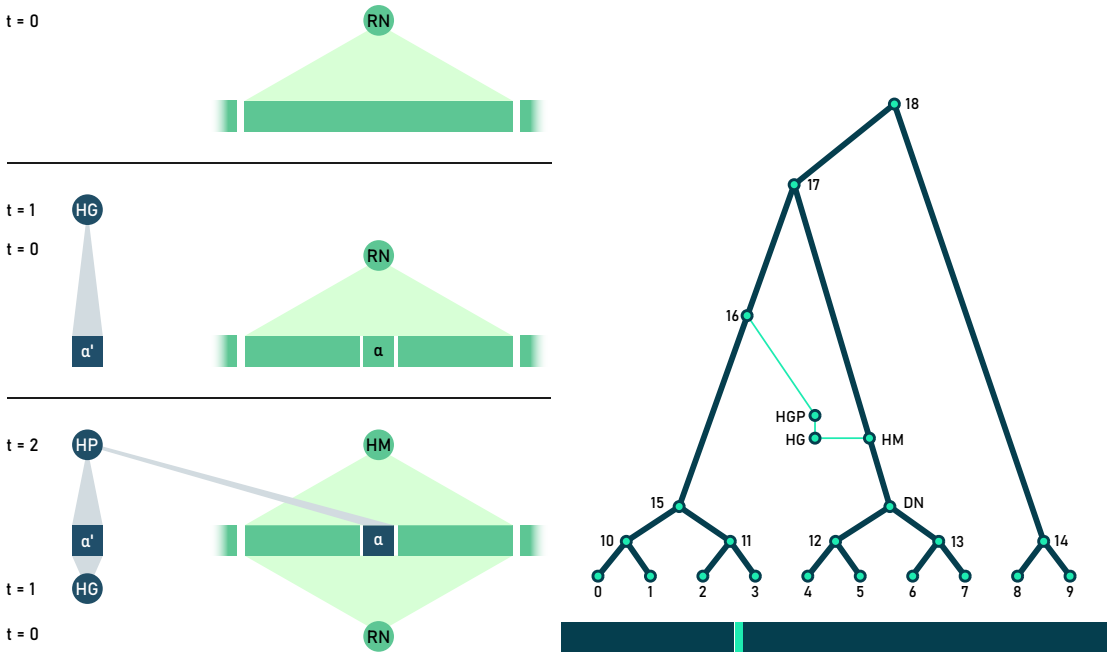


Figure 3.1: Schematics of a single HGT event divided into 3 steps (from top to bottom) (left) and in the context of a complete tree (right).

In order to represent the gene transfer event, and to be able to place mutations, edges are introduced (Step III in 3.1). These edges span from:

- `HM` to `RN` over the whole line (segment chain).
- `HP` to `HM` over the `alpha` region.
- `HP` to `HG` over the `alpha` region.

Due to the strict structural integrity checks of *tskit*, the `HP`–`HM` edge is appended to a separate edge buffer and are merged in the latter described mutation placing algorithm.

Given the potential for new HGT lineages to emerge at a faster rate than the occurrence of common ancestor events, there is a risk of divergence in the simulation. To mitigate this, we set the HGT rate to zero once the elapsed (simulated) time exceeds five times the time expected in the absence of HGT events. This ensures that no new HGT lineages are generated. The time to the next coalescent event is modelled using an exponential distribution with parameter $\lambda = n(n-1)$, where the n corresponds to the current number of lineages at the given time step (see 2.2.2).

Given that the expected value of an exponential distribution, and thus the time to the next coalescent event, is $1/\lambda$, the expected time for all lineages to coalesce - assuming no HGT, gene conversion, or other lineage-modifying events - is denoted as, $\mathbb{E}_{\text{time}}$ and calculated as follows:

$$\begin{aligned}
\mathbb{E}_{\text{time}} &= \sum_{i=2}^{i=n} \frac{1}{i(i-1)} \\
&= \sum_{i=2}^{i=n} \frac{1}{i-1} - \frac{1}{i} \\
&= \left(\frac{1}{1} - \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{1}{3}\right) + \cdots + \left(\frac{1}{n-1} - \frac{1}{n}\right) \\
&\quad \text{In a telescoping series, intermediate terms cancel out,} \\
&\quad \text{leaving only the first term of the first and last term oft the last fraction.} \\
&= 1 - \frac{1}{n}
\end{aligned} \tag{3.3}$$

We therefore use $5 \times \mathbb{E}_{\text{time}} = 5 - \frac{5}{n}$ is as limit.

3.3 Mutation Simulation

An adaptation of the regular `sim_mutation` function from *msprime* is necessary as the horizontal gene transfer tree structures cannot be represented as a `tskit.TreeSequence` in the current version of *tskit*. This restriction is due to *tskit's* rule that each node may have only one parent per position, as described in the previous section. Furthermore, the mutation algorithm lacks the ability to detect mutations occurring on HGT edges and their interactions with regular mutations.

The structure of the adapted mutation simulation is based on the original implementation and proceeds as follows:

1. **Placement of mutations**: In this step, mutations are assigned to each edge. The number of mutations is drawn from a Poisson distribution with a mean based on their length, which represents the time between parent and child, and span, which is the distance from left to right boundaries. This step randomly determines the location within these boundaries for a mutation to occur, noting only the presence of a mutation without specifying its type or any other information.

2. **Tree traversal**: For each position in the genome, this process records the corresponding tree structure. It traces the edges from the root node responsible for that position (`edge.left <= bp_l and bp_r <= edge.right`) to a leaf, documenting any parent-child relationships along the way. Since the genome is divided into segments, and the tree is identical for each position in that segment, it is sufficient to compute this tree structure for each segment, which improves efficiency.

3. **Apply mutations**: The first mutation, which is the oldest in time, is applied based on the mutation model. For the subsequent mutation, the tree, which was recorded in the previous step, is traversed upwards to identify the parent mutation. This mutation could be on a different edge. For each edge the last mutation is stored in a hashmap for quick access. In the case of two or more parental mutations, due to multiple incoming edges (some of which are HGT edges), a check is made to see if one represents a gene gain. If so, this mutation is propagated and used as the current parental state. The other mutation is discarded. A flag is stored as metadata to mark the mutation if it was affected by a HGT event. To save memory, this metadata is stored as unsigned 8-bit integer, using the second least significant bit as HGT flag. After determining the derived state, the hashmap is updated with the current mutation and the next one is processed.

4. **Sentinel Mutations**: To ensure that the final state of each position is correctly calculated and, that HGT edges are not ignored, sentinel mutations are inserted at time 0. These sentinel mutations are silent, i.e. do not change the state, but ensure that downstream algorithms, such as the genotype matrix and the GFS calculation, work correctly, as they only take the last mutation into account. These mutations are marked by the least significant bit of the metadata flag.

5. **Fill the tables and remove the HGT edges**: This final step involves filling the mutation table and removing the HGT edges to produce a valid `tskit.TreeSequence`.

The resulting tree, now validated and embedded with mutations, is returned.

3.4 Events on a fixed tree

To test the effects of horizontal gene transfer or gene conversion and estimate the parameters using real-world data, the simulation must follow a specified tree structure. This feature is introduced in this chapter. The objective is to maintain the provided tree structure while allowing regular simulation events, such as gene conversion, to occur. The structure can be specified either by a Newick string or derived from a `tskit.TreeSequence` object.

Several structural changes were required to incorporate this functionality:

- **Segment class**: The `Segment` class is extended by a new attribute called `origin`. This attribute will be a set that keeps track of the lineage IDs from which the current segment originates. Initially, this set contains only the ID of the leaf from which the segment originated. During coalescence events, the origin sets of the merging segments are combined.
- **Newick parser**: A new parser component has been implemented to process the user-provided Newick string. This parser generates a list of tuples, where each tuple represents a coalescence event consisting of three elements:
  - The time at which the coalescence event occurs (float).
  - A set containing the IDs of the first lineage involved in the coalescence.
  - A set containing the IDs of the second lineage involved in the coalescence.

  This is archived by recursively splitting the Newick string into a left and right part until a leaf is reached.
- **TreeSequence parser**: Similar to the Newick parser, the `TreeSequence` parser generates a list of tuples by iterating through the edges. Whenever two edges point to the same parent, the time of the parent node is recorded with the IDs of the children.

The `hudson_simulate` method is adapted to handle fixed coalescence events specified in the tree. If the current simulation time (`self.t`) has exceeded the time of the next fixed coalescence event stored in the `coalescent_events` list (`self.coalescent_events[0][0]`), the algorithm will deviate from its regular behaviour. The simulation time is reset to the exact time of the coalescent event, possibly with a small epsilon value to avoid storing two events at the same time step. The method will first use the `get_emerged_from_lineage` method to retrieve all lineages that have emerged from the two ancestral lineages specified in the event. Then, for each pair of lineages (one from lineage A and one from lineage B), one lineage is randomly selected to coalesce, ensuring that the predefined tree structure is followed. For regular coalescence events (not specified in the provided tree), the method continues its usual behaviour, randomly selecting two lines to merge or skipping the event if the lines are needed later. As this feature can be directly incorporated into *msprime* a pull request was opened[2] .

[2] https://github.com/tskit-dev/msprime/pull/2276

3.5 Neutrality Test and χ^2 like statistic

In order to determine whether the observed Gene Frequency Spectrum results from a process of neutral evolution with similar gene gain / gene loss parameters or is influenced by gene conversion or HGT, we developed a statistical test. This test, which can be divided into a direct approach and a χ^2-like approach, uses both a Kolmogorov-Smirnov (KS) test and Kernel Density Estimation (KDE) to evaluate GFS patterns derived from simulating a neutral evolutionary model. KDE is mainly used to smooth initial distributions and improve results, especially when working with small numbers of simulations.

3.5.1 Direct Class Approach

In the direct method, we match each observed gene frequency class, g_i^n, with its corresponding simulated distribution under neutral evolution. These simulations are generated using the most likely parameters of a pure clonal model with gene gains and losses or the parameters of interest. A robust reference distribution is measured by simulating 1000 times.

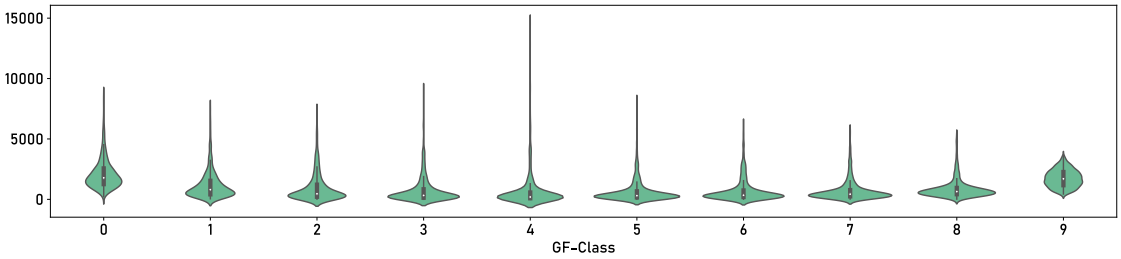


Figure 3.2: Typical distribution of gene frequency classes. 10 Individuals simulated with $\theta = 1000$ and $\rho = 0.2$.

The figure 3.2 shows a typical distribution of gene frequency classes. In this example, 10 individuals were simulated with a gene gain / gene loss rate of 1000 and 0.2 respectively. The Kolmogorov-Smirnov test quantifies the difference between the simulated distribution and the observed value. This is repeated for each gene frequency class.

Similar to the KS approach, the KDE-based direct approach fits a kernel density estimate with a Gaussian kernel to each distribution. This results in a smooth approximation of the probability density function. To evaluate how extreme the observed data point under the neutral evolution model is, we calculate a p-value by integrating the KDE from the observed data point to either positive or negative infinity. The integral represents the probability of observing a data point that is as extreme or more extreme than the observed value under neutrality.

In both the KS test and the KDE integration, the null hypothesis is that the two observed samples are drawn from the given distribution. The two-tailed alternative hypothesis states that the distributions is different.

Importantly, the p-values derived from each gene frequency class are not independent. This means that it is not possible to simply multiply them to determine the

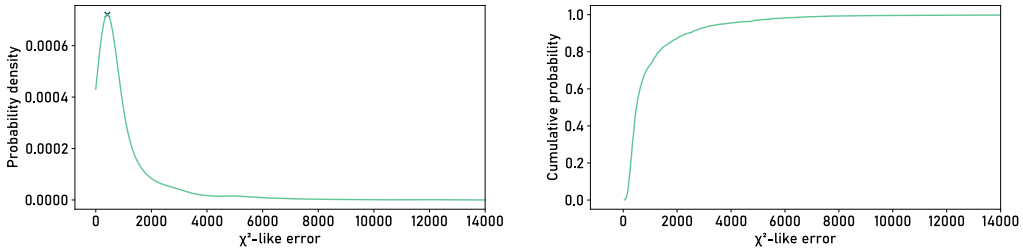


Figure 3.3: Typical Probability Density Function (left) and Cumulative Distribution Function (right) of χ^2 values.

overall significance. One solution is to use the smallest p-value, which is no longer a valid p-value in the traditional sense. It is more likely to be smaller than the significance level, even if the null hypothesis is true for all tests. This increased likelihood of Type I error is mitigated by directly implementing a Bonferroni correction:

$$p_{\text{direct}} = \min(p_{g_1^n},\ p_{g_2^n},\ ...,\ p_{g_n^n}) \cdot n \tag{3.4}$$

If this minimum p-value falls below our chosen significance threshold, we reject the null hypothesis of neutral evolution. This rejection suggests that the observed GFS could be influenced by factors other than neutral evolutionary processes, such as HGT.

3.5.2 Through a χ^2-like Summary Statistic

Through a χ^2 -like summary statistics, we can directly measure the distance between the measured gene frequency spectrum $(g_1^n, ..., g_n^n)$, and its expected values under neutral evolution. We define:

$$\chi^2(g_1^n, ..., g_n^n) = \Sigma_{i=1}^n \frac{(g_i^n - \mathbb{E}[G_i^n])^2}{\mathbb{E}[G_i^n]} \tag{3.5}$$

Following a similar approach, we generate many gene frequency spectra under a neutral evolutionary model. We then compute a χ^2-like statistics for each spectrum. Unlike previous methods, this approach produces a single distribution allowing us to directly apply a KS-test and fit a kernel density estimate. As the χ^2 value increases as the error from the expected values increases, only a one-tailed test is used.

The kernel density estimate shown in figure 3.3 characterises the Probability Density Function (PDF) of the χ^2 values. Together with the Cumulative Distribution Function (CDF), a logarithmic transformation of it shows similarity to a normal distribution, although its shape is slightly skewed, indicating that the χ^2 values follow a log-normal distribution.

To account for the variability observed between gene frequency classes, which is particularly pronounced at the boundaries (g_1^n and g_n^n) compared to the centre, we introduce a weighting parameter to control the influence of each class. This weight vector assigns relative importance to each gene frequency class. By default, in the unweighted scenario, the weight vector is defined as $(w_0, w_1, \ldots, w_n)$, where each $w_i = 1$. As these are relative weights, they must satisfy the condition $\sum_{i=0}^{n} w_i = n$. The adapted χ^2 -like summary statistic is now as follows:

$$\chi^2(g_1^n, ..., g_n^n) = \Sigma_{i=1}^{n} \frac{(g_i^n - \mathbb{E}[G_i^n])^2}{\mathbb{E}[G_i^n]} w_i \tag{3.6}$$

3.6 Minimal Site Count

While investigating the effect of the number of sites on the simulation runtime, it became clear that a low number of sites is desirable in order to run multiple simulations efficiently and to obtain reliable neutrality test results. However, in the infinite gene model, a high number of sites, ideally approaching infinity, is necessary as otherwise a gene gain event is likely to occur at a site where a gene already exists. This includes gene gain events where the root ancestral state is 'present'. It is therefore desirable to find the minimum number of sites that will not significantly bias the resulting statistics. An analytical and a computational approach have been used to do this. For a single event the probability that a given event is a gene-gain is:

$$P_\theta = \frac{\theta}{\theta + \rho \cdot s}$$

with s the total number of sites. With that, the probability that at least one the s sites is hit twice is:

$$P_{\text{double}} = 1 - (1 - P_\theta^2)^s = 1 - (1 - (\frac{\theta}{\theta + \rho \cdot s})^2)^s \tag{3.7}$$

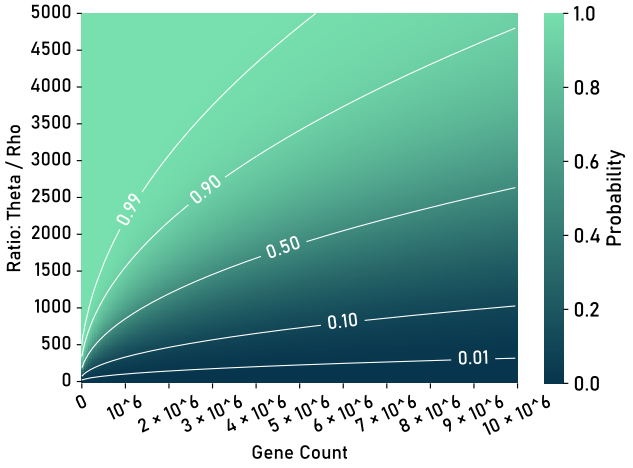


Figure 3.4: Probability that at least one double gene gain event is happening.

The figure 3.4 showed that double gene gain events remain likely, even for small gain/loss ratios, unless a very high number of sites ($>> 10^6$) is used. The second approach looked at the effect on the χ^2-like statistics. Simulations were run with different number of sites ranging from 1000 to 500000. Each simulation ($n = 10, \theta = 100, \rho = 0.1$) was repeated 1000 times. For these simulations, the GFS was recorded and the χ^2-like statistic calculated and log-transformed. The results were compared with a reference distribution assuming a site count of 10^6, as the number of observed double mutations were low (< 10). For a low number of sites ($s < 20000$), a t-test indicated significant deviations from the expected neutral model, as indicated by the very low p-values. For higher s, the p-values increased and exceeded the 0.05 threshold, indicating less deviation from the expected neutrality. The table with all p-values can be found in appendix 5.1. We therefore concluded that double gene gain can significantly bias the χ^2-like statistical summary.

Despite these observations, a hard limit on the number of sites has not been implemented in order to maintain flexibility for the user. Nevertheless, the number of double gain events is tracked to ensure the reliability of the results. A warning is given if the frequency of these events exceeds 1% of the total number of mutation events.

3.6.1 Relocation of mutations

Apart from as increasing the total number of sites, an alternative strategy is to relocate the double gene gain mutations. In the case of recombination or gene conversion, different trees describe different segments of the genome. Therefore, moving a mutation to an unused position could place it on a different tree and change its properties. This is because mutations are more likely to occur twice in the same place on longer trees. Consequently, the new position must be selected with the same bias, making random repositioning impractical. Although this could be assessed by repositioning the mutation only within the bounds, this would drastically limit the number of mutations that could be repositioned. Therefore, our implementation of random repositioning focuses only on scenarios without recombination or gene conversion. This process involves extracting the tabular representation of the tree and creating a mask for double gene gain mutations based on the derived and parental states. These identified mutations are then duplicated and mapped to a new, unused position and site. To optimise computational efficiency, we access the data directly and create an array of all mutations / sites, bypassing the resource-intensive creation of `MutationTableRow` and `SiteTableRow` objects. While this is a fast and intuitive approach, it is limited by the number of possible new positions. Thus, a `RuntimeError` is raised if the number of double mutations exceeds the number of free sites.

3.6.2 Genome Splitting

An alternative approach to dealing with double gene gain events is to simulate with twice the number of sites. This will double the number of mutations and approximately double the simulation time. Following the simulation, we divide the mutations into a 'left' and a 'right' partition, ensuring an equal number of sites in each. In the left partition, we eliminate all double gain mutations (i.e., gain mutations with gain mutations or 'present' ancestral states as their parents). On the right, we remove mutations that are not double gain mutations. In addition, we reset the ancestral state drawn from the root distribution to 'absent' to avoid sites marked as 'present' without mutations.

Finally, we use the `tables.compute_mutation_parents()` function to correct any missing parental relationships to ensure the integrity of the tree structure. This allows us to reconstruct the tree without double mutations and similar features.

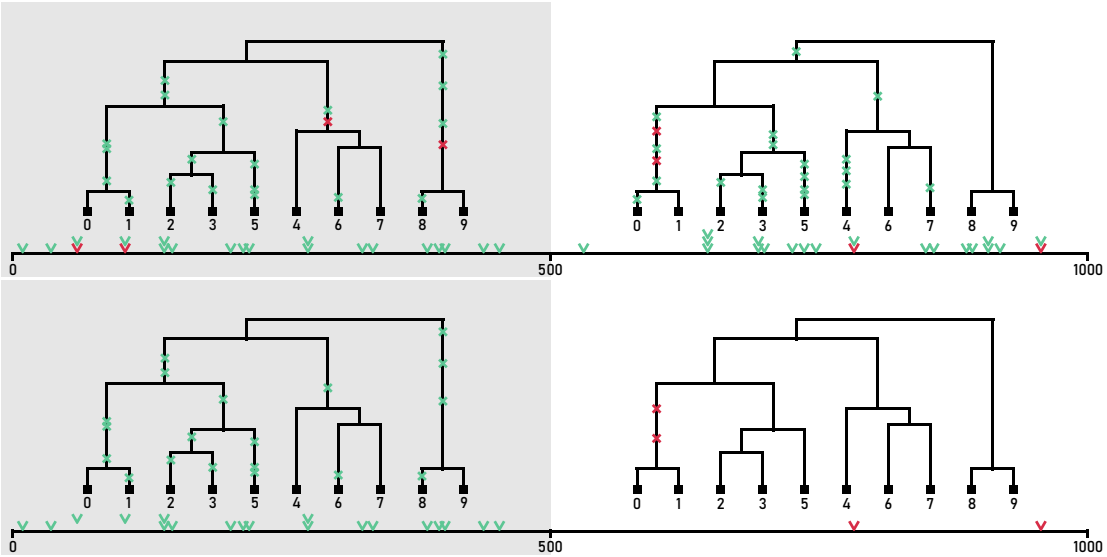


Figure 3.5: Tree with mutations before (top) and after (bottom) splitting and removal of mutations.

If, as in this example, a genome with 500 sites were desired, a simulation with 1000 sites would be used. Here 42 mutations were simulated, 4 of which were double mutations. On the left partition of the genome, all (2) double mutations were removed, while on the right partition only the double mutations (red) were kept. While the genome has 1000 sites, it has the same number of present genes as a simulation with 500 under the IMG model.

3.7 Integration

The gene gain and loss mutation model has been integrated with the customised HGT ancestry and mutation simulation model to form a complete framework. To provide a user-friendly interface, mutation relocation and tree fixation have also been added to this function. Care had to be taken when relocating mutations as some sentinel mutations would be identified as double gene gain mutations. An

additional mask using the mutations' metadata has been introduced to account for this. Due to its superior performance, the C implementation is used over the Python version whenever possible, especially when $\gamma = 0$ zero or when no HGT events have been simulated.

	Parameter	Type	Description
θ	`theta`	`int`	Gene gain rate.
ρ	`rho`	`float`	Gene loss rate.
	`ts`	`tskit.TreeSequence`	`TreeSequence` to simulate gene gain and loss directly on this tree.
	`hgt_edges`	`List[tskit.Edge]`	List of HGT edges that can't be represented in the `TreeSequence`.
n	`num_samples`	`int`	Number of samples.
s	`num_sites`	`int`	Number of sites in the genome.
κ	`gene_conversion_rate`	`float`	Gene conversion rate.
	`recombination_rate`	`int`	Recombination rate.
ρ	`hgt_rate`	`float`	Horizontal Gene Transfer rate.
	`ce_from_nwk`	`str`	Tree structure in Newick format to follow during simulation.
	`ce_from_ts`	`tskit.TreeSequence`	Tree structure in `TreeSequence` format to follow during simulation.
	`check_double_gene_gain`	`bool`	Warning if more than 1% of all events are double gene gain events.
	`double_site_relocation`	`bool`	Simulates with 2× `num_sites` and fixes double gain mutations.
	`relocate_double_gene_gain`	`bool`	Repositions double gene gain mutations.

Table 3.1: Parameters of the `gene_model` function that combines the HGT ancestry simulation with the gene gain / loss model.

3.8 Estimating parameters for real world pangenomes

To test the models' capabilities, we used real-world pangenomes, calculated the GFS and optimised the simulation parameters using a *Simplicial Homology Global Optimisation* and *Differential Evolution* approach to match the simulated GFS with the real ones. Samples of multiple bacterial species we downloaded and processed by Hannah Götsch using *PanX*.

PanX is a software tool designed to streamline the construction and analysis of pan-genomes [DBN17]. It begins by using the *DIAMOND* alignment algorithm to perform an all-against-all similarity search, identifying pairs of genes across the input genomes that have significant sequence similarity [BXH14]. These similarity results are then clustered using a Markov clustering algorithm, which groups genes that are likely to have a common evolutionary origin (orthologs). As *DIAMOND* focuses on proteins, *PanX* has a separate process to identify and cluster non-coding genes such as ribosomal RNAs (rRNAs) using BLASTn. To cope with the potential computational burden of analysing thousands of genomes, *PanX* avoids direct all-against-all comparisons. Instead, it clusters smaller subsets of genomes, then represents these clusters by 'pseudo-genomes' and repeats this process hierarchically. This strategy helps to maintain a near-linear growth in computational cost as the number of genomes increases.

PanX constructs phylogenetic trees for each gene cluster using tools such as MAFFT (for alignment) and FastTree (for tree-building) [Kat02] [PDA10]. These trees are carefully analysed to split gene clusters that may contain paralogs (genes related by duplication within a genome). *PanX* uses a combination of branch length thresholds and a 'paralogy score' to finely separate orthologs into distinct groups. It also has a process to address genes that may have been missed in

the initial clustering, combining them based on sequence length and phylogenetic analysis.

Using alignments of core genes (present in all strains), *PanX* builds a 'core genome' phylogenetic tree using *FastTree* and refines it using *RaxML*. This tree shows the overall evolutionary relationships between the strains. Later, this tree is used to remove potential oversampling using *PhyloThin*.

The thinned core genome trees and gene presence/absence matrices were exported and used for subsequent parameter estimation using the Simplicial Homology Global Optimisation algorithm.

The Simplicial Homology Global Optimisation (SHGO) algorithm is a method for efficiently identifying local minima in objective functions, especially when, as in our case, the function is costly to evaluate [ESF18]. For this we use the implementation of the *SciPy* package [Vir+20]. As objective function we used the χ^2 error of the mean GFS of 24 simulations against the true GFS. This choice was a compromise between run time and variability of the error. The algorithm uses concepts of simplicial integral homology and combinatorial topology to navigate complex problem landscapes, which makes it useful for high-dimensional and complex applications such as exploring energy landscapes or, in our case, the χ^2-like error for different parameters. Of course this step assumes that the phylogenetic tree, computed by *PanX* is correct.

SHGO works by starting with the construction of a simplicial complex from the feasible search space. This complex forms a topological abstraction where the vertices represent potential solutions and the simplicial links illustrate possible optimisation paths. By approximating the homology groups of this complex, SHGO is able to map the structure of the search space and pinpoint areas likely to contain local minima. The search space is defined by bounds that we set to:

Parameter	start	stop	Description
θ	0.1	average number of genes $\times 3$	Gene gain rate.
ρ	0.0001	number of samples	Gene loss rate.
κ	0	0.01	Gene conversion rate.
γ	0	0.01	Horizontal Gene Transfer rate.
recomb	0	0.01	Recombination rate.

Table 3.2: Parameter search-space.

based on our previous experience and runtime feasibility.

The algorithm uses intelligent sampling methods based on the vertices of the simplicial complex, guided by Sperner's lemma. This lemma helps to approximate locally convex subdomains that are likely locations of local minima, significantly reducing the number of function evaluations required.

The other optimisation technique we used is the Differential Evolution (DE) approach by Storn and Price [SP97] which finds an optimal solution by evolving a population of candidate solutions over time. Differential Evolution involves four main steps: initialization, mutation, crossover, and selection. Initially, a population of 15 candidate solutions is randomly generated, within our search space. We use the same bounds as in the SHGO approach. Each candidate solution,

also called an individual, is represented as a vector. Once the initial population is established, the algorithm proceeds to the mutation step. In this phase, for each candidate solution in the current population, a mutant vector is generated. This is done by selecting three distinct individuals from the population and combining them by adding the weighted difference between two of these individuals to the third one. The mutation is followed by the crossover step which aims to increase the diversity of the population by randomly combining components of the mutant vector with the candidate vector to create a trial vector.

Finally, in the selection step, the trial vector is compared to the target vector based on their fitness values, which are determined by the objective function being minimised. If the trial vector has a better fitness value than the candidate vector, it replaces it in the population. This ensures that the population evolves towards better solutions over successive generations.

Once the minimisation process had converged, the parameters were recorded and further refined using a Downhill Simplex (DS) approach. This additional step is necessary as both SHGO and DE algorithms excel at exploring the wider χ^2 value landscape, but struggle in the final, local optimisation.

For this step, the number of simulations per evaluation of the objective function was increased from 24 to 48 in order to reduce the variability of the χ^2 like value. The result of this process is a list representing several gene frequency spectra, providing a distribution and χ^2 like measurement indicating the fit of the simulated parameters to the true parameters.

Chapter 4: Results and Discussion

4.1 Effect of Gene Conversion on the GFS

The effect of gene conversion on the gene frequency spectrum becomes most apparent when simulations are performed on a fixed tree with a recognisable GFS pattern. In our constructed tree (figure 4.1) with three distinct clades (C1: individuals 0-3, C2: individuals 4-7, C3: individuals 8-9), the two leftmost clades consist of four individuals each, while the right clade consists of only two. Due to numerous gene gain events along the long branch from node 18 to 17, we expect these genes to spread across the entire left side, resulting in a GFS peak at class 8. Similarly, along the branches 17 to 15 and 17 to 16, the gene gain affects the entire clade, resulting in a GFS peak at class 4. The same principle applies to the 18 to 14 branch, but for the smaller clade, resulting in a peak at class 2. Genes present in all 10 individuals are less likely to stem from regular event as this would require at least two gene gain events at the same site on different branches, but rather stem from the ancestral state. As the GFS includes those and does not distinguish between those genes we expect another peak at class 10. While it is easy to infer the GFS from a tree structure, it is not possible to do the inverse.

The introduction of a single gene conversion event causes a gene in a few individuals to follow an alternative phylogenetic tree. This is illustrated by the light green segment / branches in the figure that oppose to the dominant tree in dark blue. Due to the long edge from 18 top GC1 an increase of the peak Gene Frequency Class 6 would be expected.

As gene conversion increases, more genome segments are determined by different trees, disrupting the effect of the initial structure. Consequently, the expected GFS across different fixed trees with gene conversion converges towards the expected GFS of random trees under neutral evolution. Nevertheless, certain features persist even with increased gene conversion rates, such as the peak at class 4 and class 10. These would require numerous gene conversion events to counteract the strong influence of long edges and the root probability of the ancestral state. This is visible in figure 4.1 (right) which depict the GFS for different gene conversion rates and the mean χ^2-like error.

When inspecting the non-normalized GFS of simulations with a fixed tree, it apparently it does not exactly match the expected GFS of neutral random trees. Instead, it reflects its shape but is slightly shifted based on the gene gain/loss parameters. This shift is caused by the emergence of new lineages due to gene conversion, which increases the time to coalescence for all lineages. As a result, the total potential for gene gain / loss events is increased. The χ^2-like error between the mean GFS and the expected GFS highlights this discrepancy. Initially, the error drops to a low level but never reaches zero.

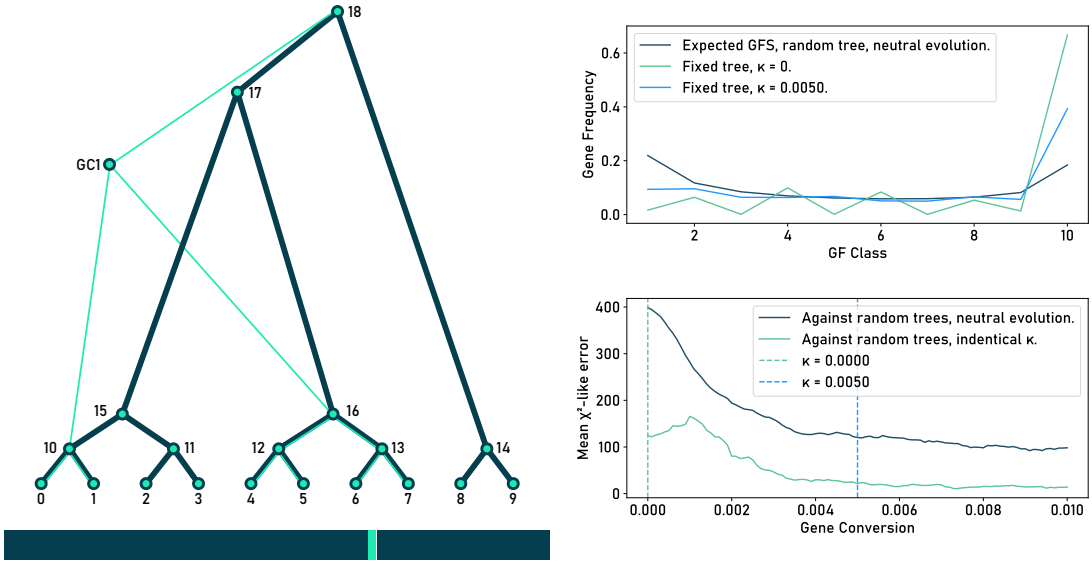


Figure 4.1: Tree with one gene conversion event (left), its GFS for different gene conversion rates (top right) and its respective χ^2-like error (bottom right).

4.2 Effect of HGT on the GFS

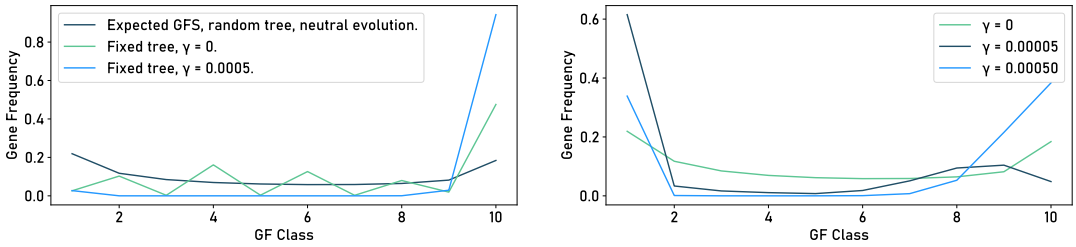


Figure 4.2: GFS for different HGT rates (left) on a fixed tree and on random trees (right).

A high value of γ shifts the gene frequency spectrum to the right, indicating that most genes are present at high frequency resulting in a characteristic J shape. This means that when new individuals are sequenced, it is unlikely that new genes will be discovered that were not previously observed. When a fixed tree is used, a similar effect to gene conversion is expected. Along with the loss of characteristic features of the simulated tree, a shift to the right is expected, resulting in a smaller peak at lower gene frequency classes. This is due to new edges introduced by horizontal gene transfer events changing the original tree structure. This effect is visible in figure 4.2.

The increase in gene frequency is due to the nature of HGT, where any gene acquisition in a lineage, whether by vertical inheritance or HGT, is counted as a gain. This effect is clearly visible in figure 4.2 for simulations without tree fixation.

4.3 Neutrality test robustness

The properties of the neutrality test differ between the χ^2 approach and the direct approach. Understanding these differences helps to interpret test results and to determine the situations in which each approach may be most informative.

By relying on the minimum p-value across all GFS classes, the direct approach is sensitive to small changes within individual classes. Small variations in the underlying parameters (θ, ρ, γ and κ) can lead to pronounced changes, especially at the edges of the GFS in classes 1 and n. The direct approach also shows reduced sensitivity to overall upward or downward shifts in the GFS, as these large changes may not significantly alter individual class probabilities. These changes are captured by the χ^2 approach by summing the respective errors.

4.3.1 With neutral evolution as reference

We assessed the robustness of our model by calculating the mean likelihood values and 90% percentiles obtained from multiple neutrality tests. This provides a measure of the stability of the test under different assumptions about evolution. A reference distribution was generated by sampling the GFS of 5000 simulations under neutral parameters. Then 25 simulations were run for each step in our list of desired parameters (table 5.2, figure 4.3).

The plots of these results include a vertical blue line indicating the true parameters used in the neutral reference simulation:

- **Gene Gain** (θ): Tests around theta perform consistently well, forming the expected bell curve around the true parameter values. Deviations as large as 1000 lead to significant drops in p-values (below 0.1).

- **Gene Loss** (ρ): A similar pattern can be observed for Rho. Here we see a saturation effect, as at high gene loss values the effect is less pronounced, as most events occur at sites where no gene is already present.

- **Gene Conversion** (κ): As higher gene conversion rates shift the gene frequency spectrum towards that expected from random tree models, the variance of the GFS decreases. As a result, the tests do not detect deviations due to increased gene conversion rates. However, if a simulation with gene conversion ($\kappa = 2$) is used as a reference, it is possible to detect values originating from a simulation with low gene conversion.

- **HGT** (γ): The effect of HGT is clearly visible, with a half bell curve around the true value. As the effect of HGT is strong on individual GF classes as well as all classes together, it is well picked up by the χ^2 based and direct test

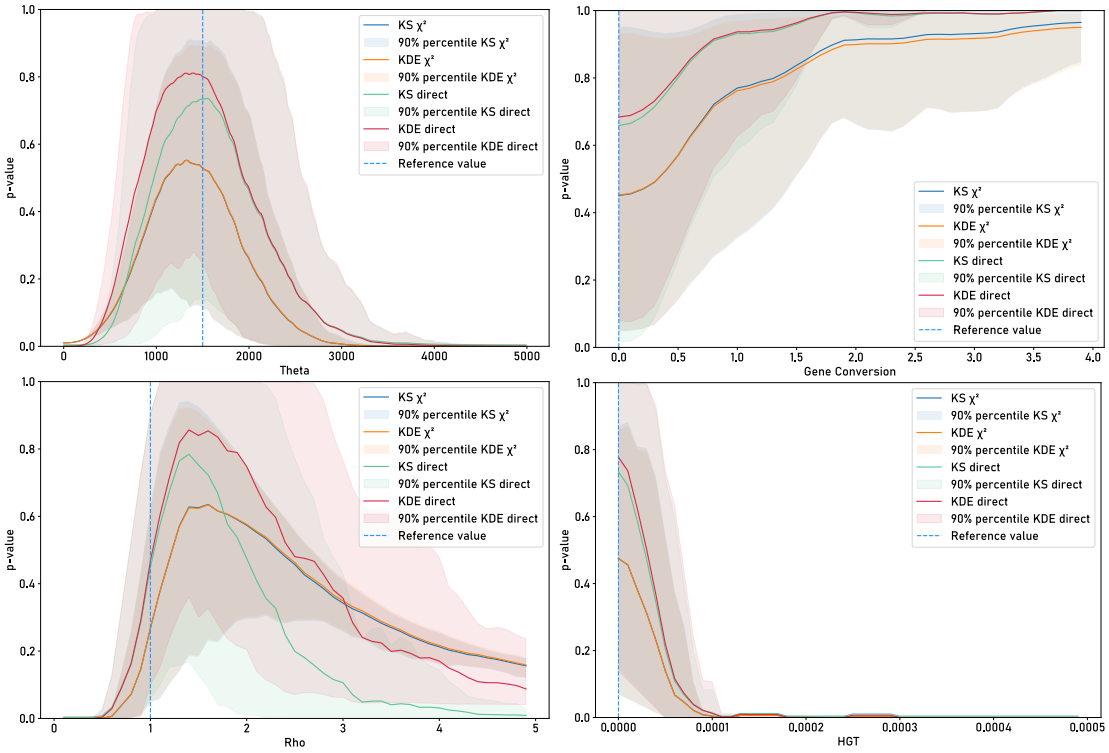


Figure 4.3: Robustness of the neutrality test for different θ (top left), ρ (bottom left), κ (top right), γ (bottom right) values with a neutral evolution as reference.

4.3.2 Non-neutral evolution as reference

As the neutrality test is based on a simulated reference distribution it is also possible to test against an evolution with gene conversion or HGT. When considering non-neutral evolutionary processes, the expected gene frequency spectrum distribution must be adjusted accordingly. In the following test, the reference distribution was obtained by simulating with $\gamma = 0.0001$. A total of 5000 samples were recorded. Similar characteristics are observed for changes in θ and ρ: a bell curve around the true θ value and a tail for ρ due to the saturation effect. See figure 4.4. The difference between the KDE-smoothed and KS-based χ^2 test appears to be due to rare outliers. These outliers affect the smoothing process, resulting in a wider distribution and therefore a less effective test. For HGT, the test is able to distinguish whether the new data stems from a simulation with less or no HGT, and shows a right tailed bell curve. This left part of the curve reflects the test results from the neutral evolution, as it is the inverted version if it. For higher HGT values, the sensitivity decreases and significantly higher HGT values are required. The reason for this is that the reference HGT value of 0.0001 already has a strong effect on the gene frequency spectrum, and the relative change for higher values is small as the effect saturates. A similar pattern was observed for simulations with gene conversion ($\kappa = 2$, figure 4.4 bottom right) as a reference. Here the test is able to detect GFS from the simulation with less gene conversion. Detection of higher values of κ is not possible, as the effect of gene conversion saturates after pushing the GFS to the expected GFS for random trees.

The complete list of measurement can be found in appendix 5.1.

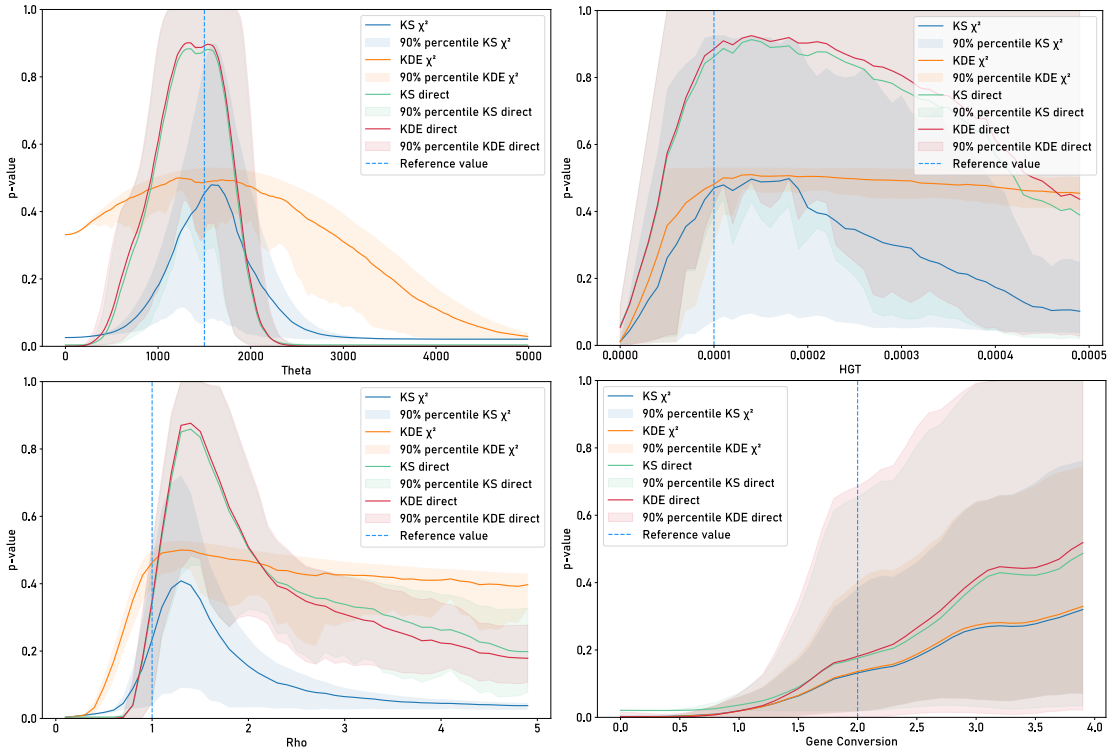


Figure 4.4: Robustness of the neutrality test for different θ (top left), ρ (bottom left), κ (top right), γ (bottom right) values with $\gamma = 0.001$ (top / bottom left, top right) and $\kappa = 2$ (bottom right) as reference.

4.4 Runtime

To ensure that the new gene model has a feasible runtime - necessary due to the multiple runs required for neutrality estimation - we investigated the effect of different parameters.

Simulations without HGT are using the underlying C implementation, while those with HGT are using a pure Python implementation. This difference is expected to lead to significant differences in runtime. To capture this effect, we timed a wide range of values for the parameters n, s, θ, ρ and κ with a HGT rate γ set to 0 and 0.001. Each parameter was tested in isolation, i.e. only one parameter was varied while the others were held constant at their default settings.

The parameters, their default values and their ranges tested were as follows:

Parameter	Min	Max	Step	Default	Description
n	0	10000	10	10	Number of samples.
s	1000	100000	100	10000	Number of sites.
θ	0	1000	1	100	Gene gain rate.
ρ	0.01	0.5	0.001	0.1	Gene loss rate.
κ	0	2	0.01	0	Gene conversion rate.
γ	0	0.005	0.0005	0 / 0.0001	Horizontal Gene Transfer rate.

Table 4.1: Runtime analysis parameters.

A total of 100 simulations were run for each step within a parameter's range, and run times were recorded. Some parameter combinations were also tested to identify potential pairwise dependencies. The individual plots (figure 4.5) show the mean runtimes and the 90th percentiles. Figures presenting the remaining parameters, simulation with HGT and the first difference of the runtimes are shown in the appendix 5.3. As inferred from the implementation, the runtime follows a mostly linear dependence on problem and parameter size.

- **Number of samples**: Runtime anomalies were observed for certain sample sizes, particularly around 500 for the C implementation. Initial hypotheses suggested that these anomalies might be due to default memory allocations or processor cache limitations. However, tests on different processors, regardless of L2/L3 cache size, showed similar increases at these sample sizes. Thermal throttling or switching from performance to efficiency cores was considered but ruled out due to the consistent runtime jumps at these specific points. Given the small impact of these anomalies and their absence in the Python version, they were ignored for the remainder of the study.

- **Number of genes**: The graphs show an almost linear increase in runtime as the number of genes increases, with the difference showing a slightly super-linear trend. With more genes, more positions have to be tracked, causing the increase.

- **Gene Gain** (θ): Both graphs and their derivatives show a stable linear growth in runtime with increasing theta, reflecting the additional computational load of more gene gain events and the resulting increase in the number of lines to track.

- **Gene Loss** (ρ): A lower rho value means fewer gene loss events, resulting in more lines to simulate. As rho increases, fewer lines need to be simulated due to more frequent gene losses. However, at a certain point, simulating gene loss events takes longer than the time saved by eliminating lines, leading to a linear increase in run time. To fully understand the interaction between rho and theta, we plotted the execution time over a range of their combinations and ratios.

- **Ratio of ρ to θ**: The runtime for different ratios of θ and ρ is presented in figure 5.2. The effect of ρ on runtime initially decreases execution time, followed by a linear increase as ρ increases. In this analysis, the parameter range was set from 0 to 10,000 in steps of 100 for θ and 0 to 1 in steps of 0.01 for ρ. All possible combinations were measured, and the parameter grid was shuffled to reduce the impact of sorting bias. Outliers in the heatmap are likely due to measurement methods, as there is no clear pattern.

- **Gene Conversion** (κ): This parameter has a significant effect on running time, with a more than linear increase observed. Each gene conversion event introduces a new lineage into the population, increasing the number of generations required to reach the most recent common ancestor.

- **HGT** (γ): Similarly to gene conversion, this parameter has a strong effect on run time. Each HGT event creates a new lineage that must be tracked and post-processed, leading to a superlinear increase in runtime.

Overall, these results highlight the impact of the number of lineages on the runtime and the need for a C implementation.

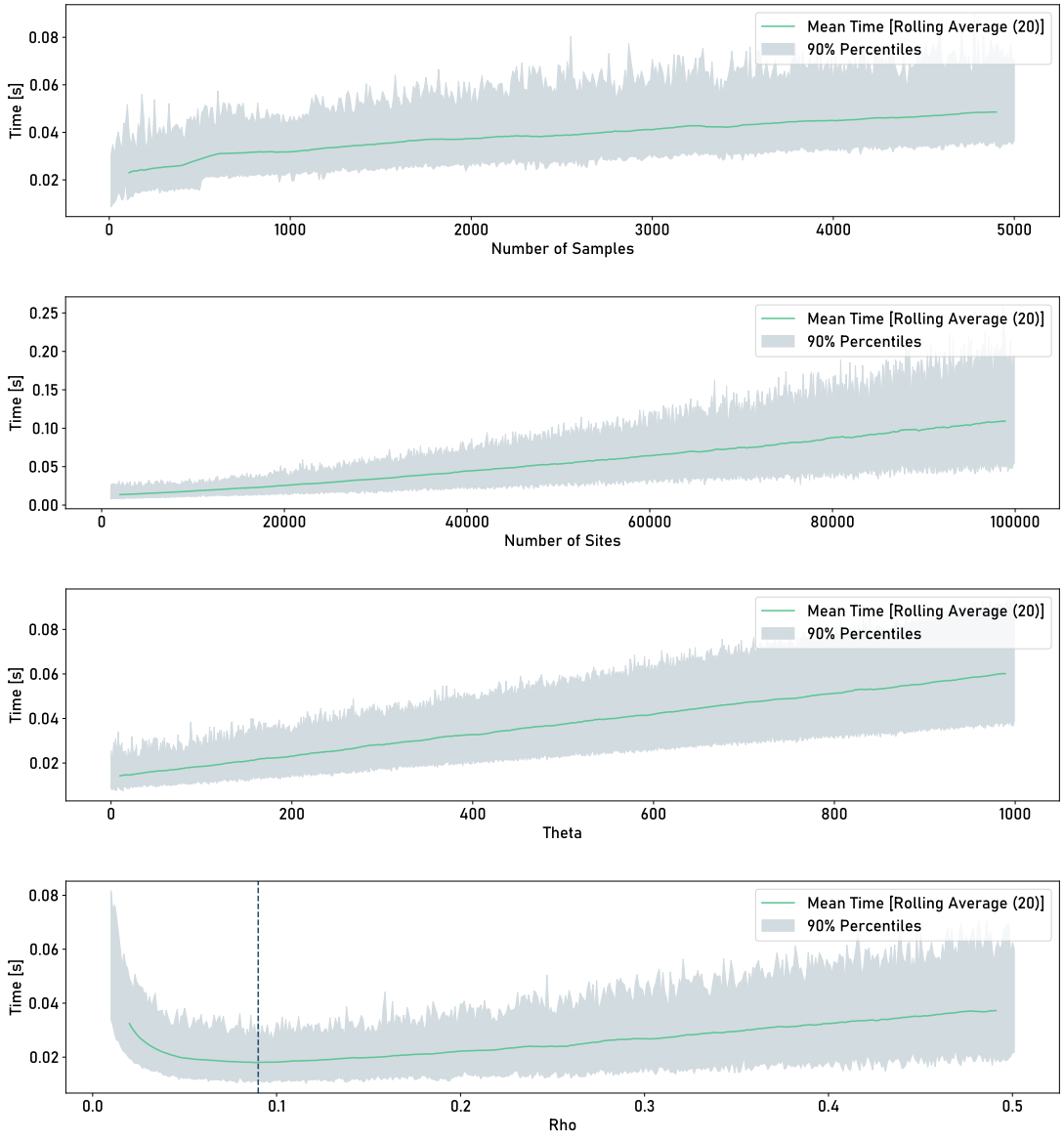


Figure 4.5: Runtime of `gene_model` for different number of samples, number of sites, θ and ρ rates for simulations without HGT.

4.5 Estimating parameters for real world pangenomes

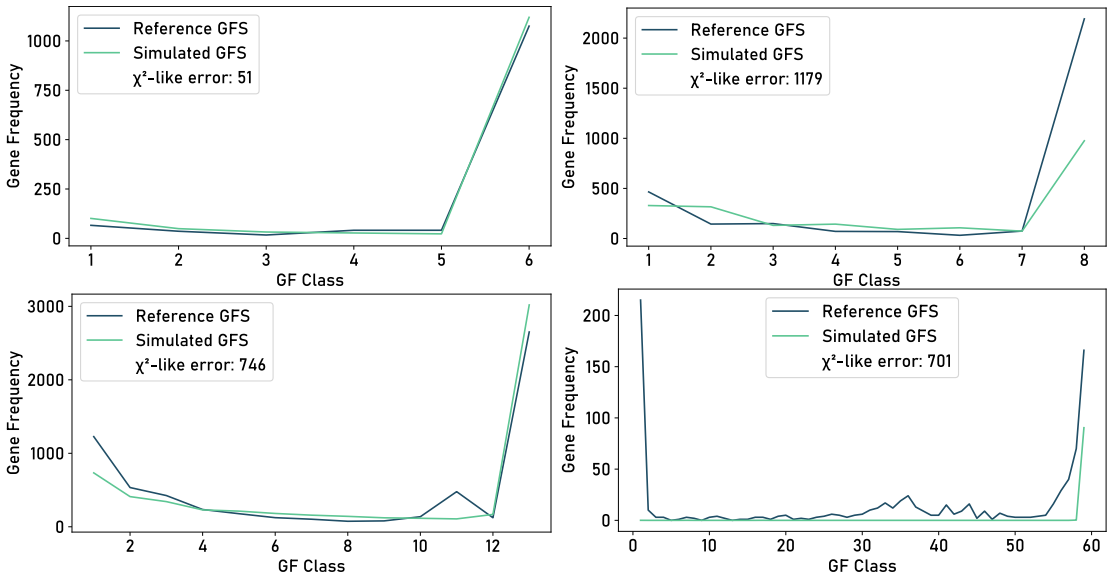


Figure 4.6: True and simulated GFS for Bartonella quintana (top left), Staphylococcus argenteus (top right), Clostridium butyricum (bottom left), Buchnera aphidicola (bottom right).

With optimised parameters, our model successfully simulated matching GFS for most pangenomes but failed for some. Although fitting is possible, as discussed in the previous section, it is slow due to the current Python-only implementation. Therefore, we only used low sample count datasets.

Bartonella quintana, chosen due to its known high horizontal gene transfer rate [QD19]. Only 6 samples were present. The fit is excellent, with a χ^2 like error of 51 and an expected high resulting HGT parameter of 0.003. This would correspond to a rate of 30 if we rescale it by the number of sites. As the used phylogenetic tree is balanced and the HGT effect is strong, a characteristic ⅃ shape is visible. An overview with all phylogenetic trees can be found in appendix 5.7.

In addition, we selected another strain with slightly more samples where a high HGT rate was expected. We chose Staphylococcus argenteus as its part of the S. aureus related complex. This strain had a similar HGT value of 0.004 The fit is acceptable with a χ^2 like error of 1170 and the significant right-shift of the GFS towards many high-frequency genes can be seen in the figure 4.6. It is important to note that the χ^2 like error is not normalised and thus can't be compared directly between samples 3.5.2.

In order to assess the model's capacity to handle the significant influence of the underlying phylogenetic tree, we selected Clostridium butyricum, which exhibits a peak in the gene frequency class 9. Despite the successful identification of parameters that resulted in matching GFS for all other classes, the optimisation algorithm was unable to identify parameters that replicated the peak in class 9. This resulted in a χ^2 line error of 749. It is our hypothesis that the optimisation

algorithm became trapped in local minima. Local minima are particularly likely to occur when testing high gene conversion or HGT rates. The strong influence of these factors on the GFS's shape can significantly improve the χ^2 like error in the first step, but then also prevents the algorithm from identifying better parameters, as further changes to other parameters will have minimal impact.

As a representative example with a low rate of HGT and a limited number of genes Buchnera aphidicola was chosen [CPM19]. Unfortunately, due to its 59 samples, it resulted in a longer optimisation time, leading to unpolished parameters with a poor score. Here the fit resulted in a χ^2 error of 701. As the optimisation process had to be aborted due to the long run time and lack of computational resources, and as the results improved over time, we are optimistic that the fit could have been significantly better.

Differential Evolution proved to be significantly better than either SHGO or simple DS for global optimisation. While DE efficiently found reasonably good parameters for the first two samples in less than 1000 steps, SHGO required over 2000 steps, leading to an impractically long runtime.

Therefore, after testing Bartonella quintana and Staphylococcus argenteus, we used DE exclusively. Locally, Downhill Simplex performs better than DE and converges to a minimum faster. Although DS does not guarantee a global minimum, the variance of the local error is larger than the error itself, blocking any further minimisation. This would be solved by using more simulations for each optimisation step to achieve a more stable χ^2 like error, which would ultimately result in an impractical runtime.

Chapter 5: Summary

This thesis investigates the dynamics of Horizontal Gene Transfer (HGT) in bacterial populations, with a particular focus on the effects on the Gene Grequency Spectrum (GFS). Traditional simulation models of genetic evolution based on clonal inheritance often overlook the effects of lateral gene flow introduced by HGT. This research aims to integrate HGT into existing models to improve their accuracy and predictive power.

The mutation model of *msprime* was extended by incorporating the concept of gene gain and loss based on the Infinitely Many Genes (IMG) model. By moving the double gene gain mutation to unused sites, the problem of having a finite number of sites that would conflict with the IMG was mitigated. This extension of *msprime* allowed the simulation of gene frequency spectra and the development of a neutrality test to assess whether a GFS results from a neutral evolutionary process.

Ancestry and mutation simulation was further improved by introducing mechanisms for tree fixation and HGT. This adaptation involved modifying the built-in ancestry simulation to create a new single-gene lineage for each HGT event, and adjusting the mutation algorithm to correctly place mutations while accounting for the potential influence of HGT. This allowed the effects of HGT on the GFS to be studied.

The model was applied to real-world pangenome data from NCBI. Parameters were optimised using Simplicial Homology Global Optimisation (SHGO) and Differential Evolution (DE) to achieve a good fit between simulated and observed genetic patterns. This was mostly achieved.

The extended *msprime* and *tskit* tools provide a robust framework for simulating and analysing genetic evolution, leading to more accurate predictions of evolutionary outcomes. The results highlight the important role of HGT in microbial evolution and show that genetic uniformity can be rapidly introduced by lateral gene flow.

However, the model has two important limitations: it assumes constant rates of gene gain and loss, which may not reflect temporal variation, and the interaction between gene conversion and HGT increases the number of lineages, resulting in an infeasible runtime for large-scale simulations. In addition, the pure Python implementation is not suitable for large-scale parameter estimation and simulation.

Future research should explore more dynamic models that account for varying rates of gene gain and loss over time, and provide a C implementation with an improved method for simulating gene conversion and HGT simultaneously. This would allow the model to be fully integrated into *msprime*.

This work has provided a framework for simulating the impact of HGT on genetic diversity in bacterial populations.

Appendix

5.1 Influence of Double Gene Gain Events

The influence of double gene gain events on low gene-count simulations with 10 samples and$\theta = 100, \rho = 0.1$. The χ^2-like values for simulations with less than 20000 genes show a significant deviation from the reference.

Number of sites	P Value	Significant
1000	2.21e-05	x
2000	8.16e-21	x
5000	3.40e-09	x
10000	8.59e-05	x
20000	0.13	
50000	0.31	
100000	0.78	
200000	0.80	
500000	0.12	

Table 5.1: Effect of double gene-gains on the log-transformed χ^2-like values for low number of sites simulations.

5.2 Neutrality Test Robustness

Parameter	start	stop	step	default
θ	0	5000	10	1500
ρ	0.1	5	0.1	1
κ	0	4	0.1	0
γ	0	0.0005	0.00001	0

Table 5.2: Neutrality test robustness parameters.

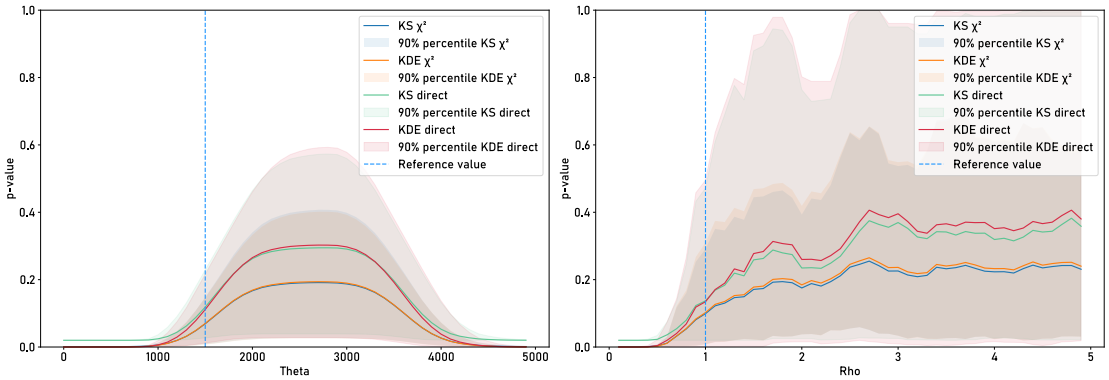


Figure 5.1: Robustness of the neutrality test for different θ (left), ρ (right) values with $\kappa = 2$ as reference.

5.3 Runtime

The first differences of the recorded runtime of `gene_model` for different parameters and the absolute runtime for different θ/ρ ratios are presented here.

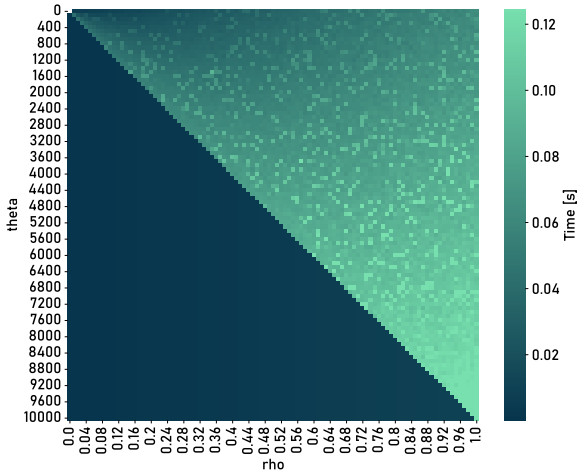


Figure 5.2: Runtime of `gene_model` for different θ/ρ ratios for 10 samples and $\kappa = \gamma = 0$.

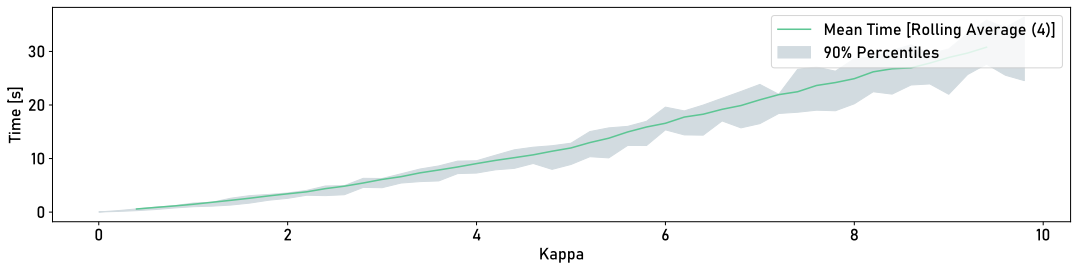


Figure 5.3: Runtime of `gene_model` for different κ rates for simulations without HGT.

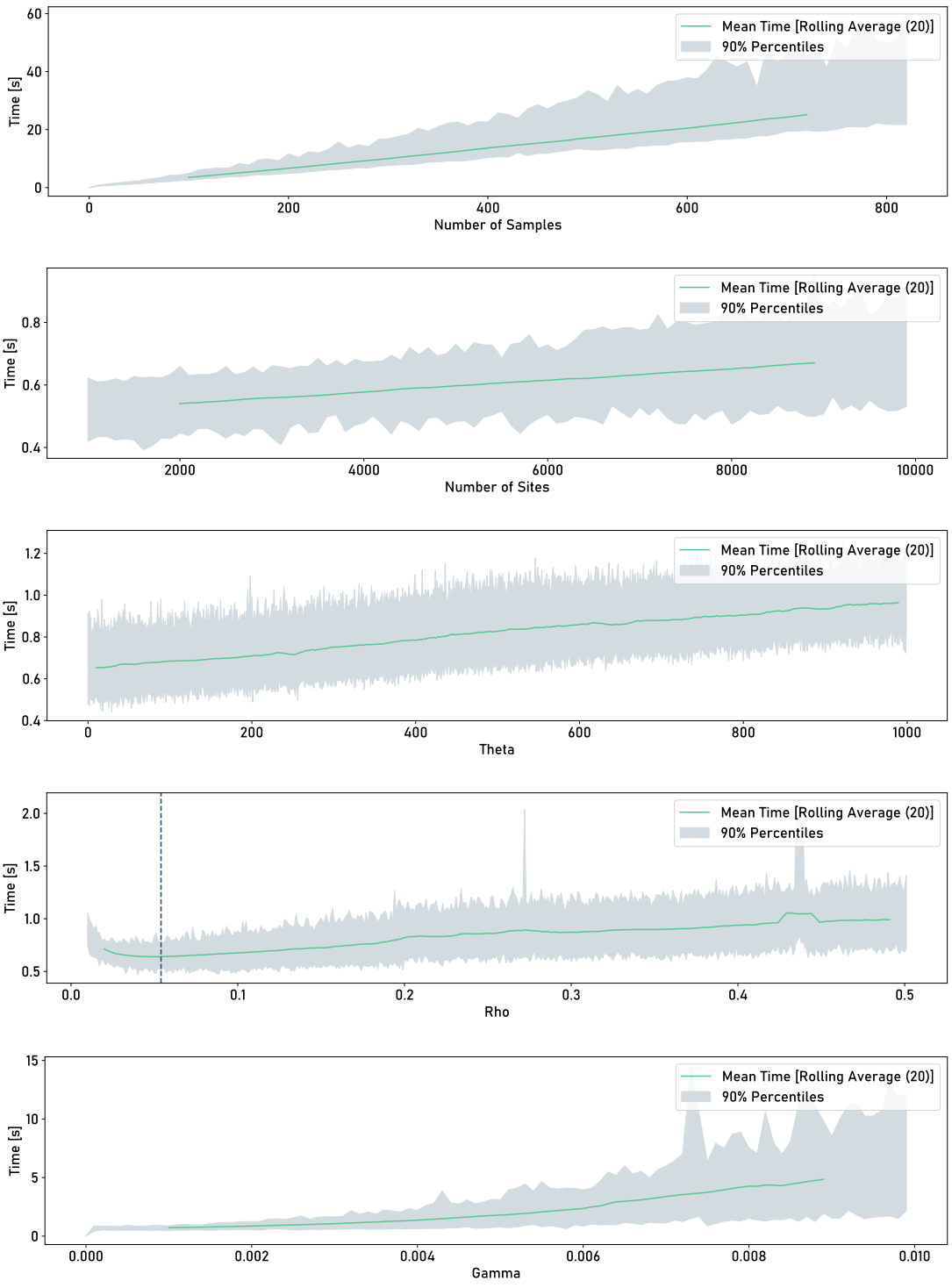


Figure 5.4: Runtime of `gene_model` for different number of samples, number of sites, θ, ρ and γ rates for simulations with HGT.

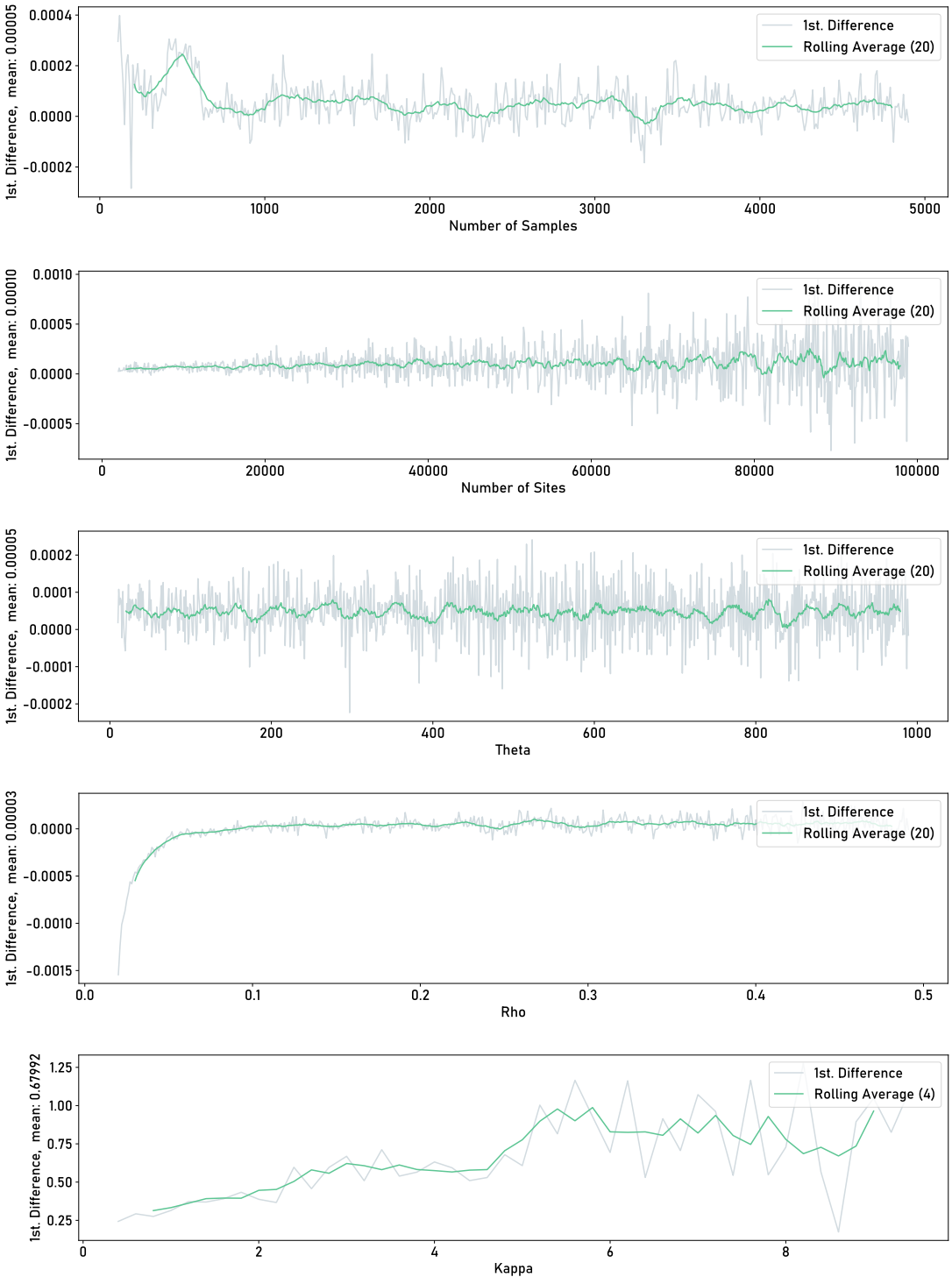


Figure 5.5: First difference of runtime of `gene_model` for different number of samples, number of sites, θ, ρ and κ rates for simulations without HGT.

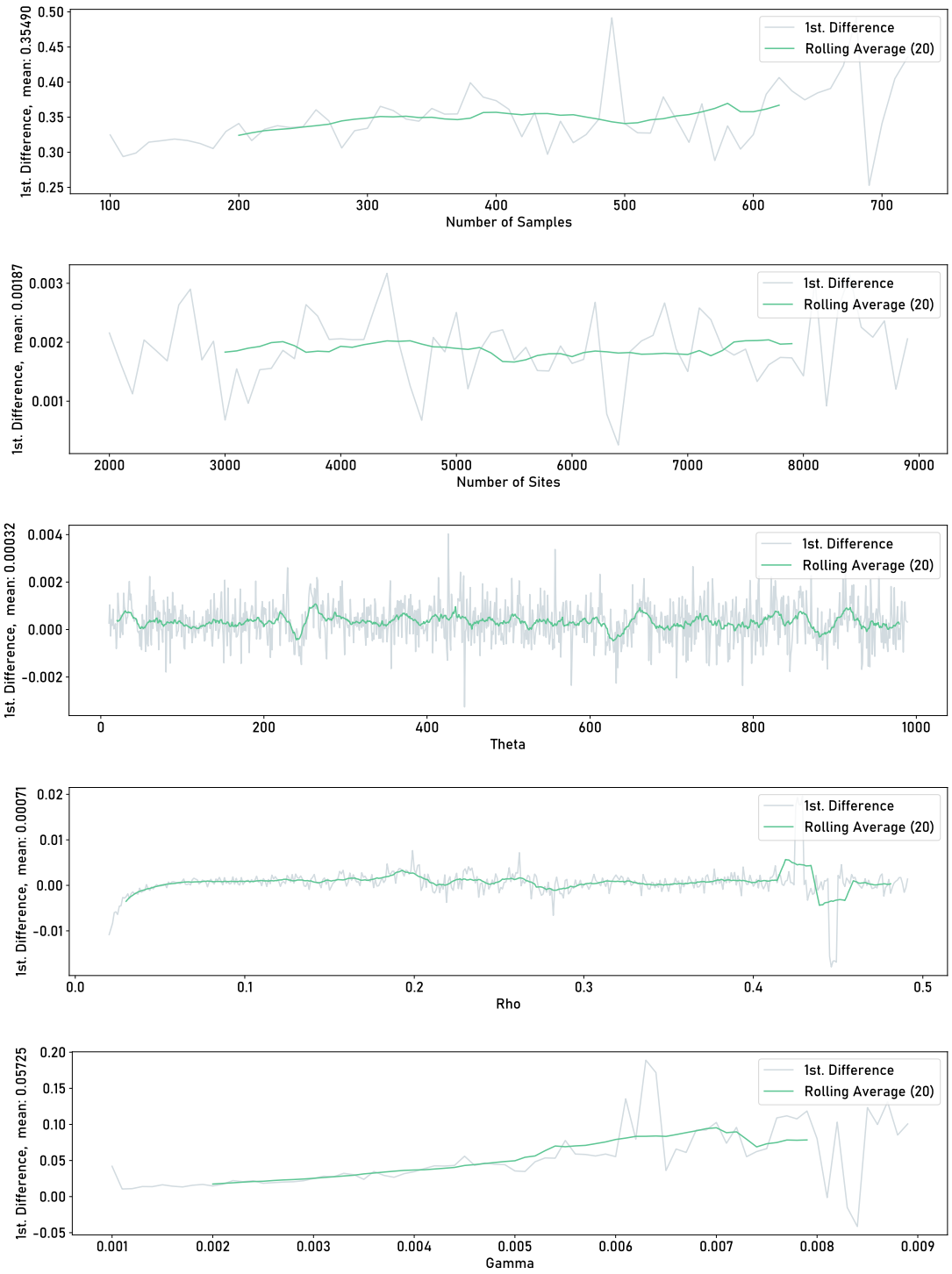


Figure 5.6: First difference of runtime of `gene_model` for different number of samples, number of sites, θ, ρ and γ rates for simulations with HGT.

5.4 Trees for parameter estimation

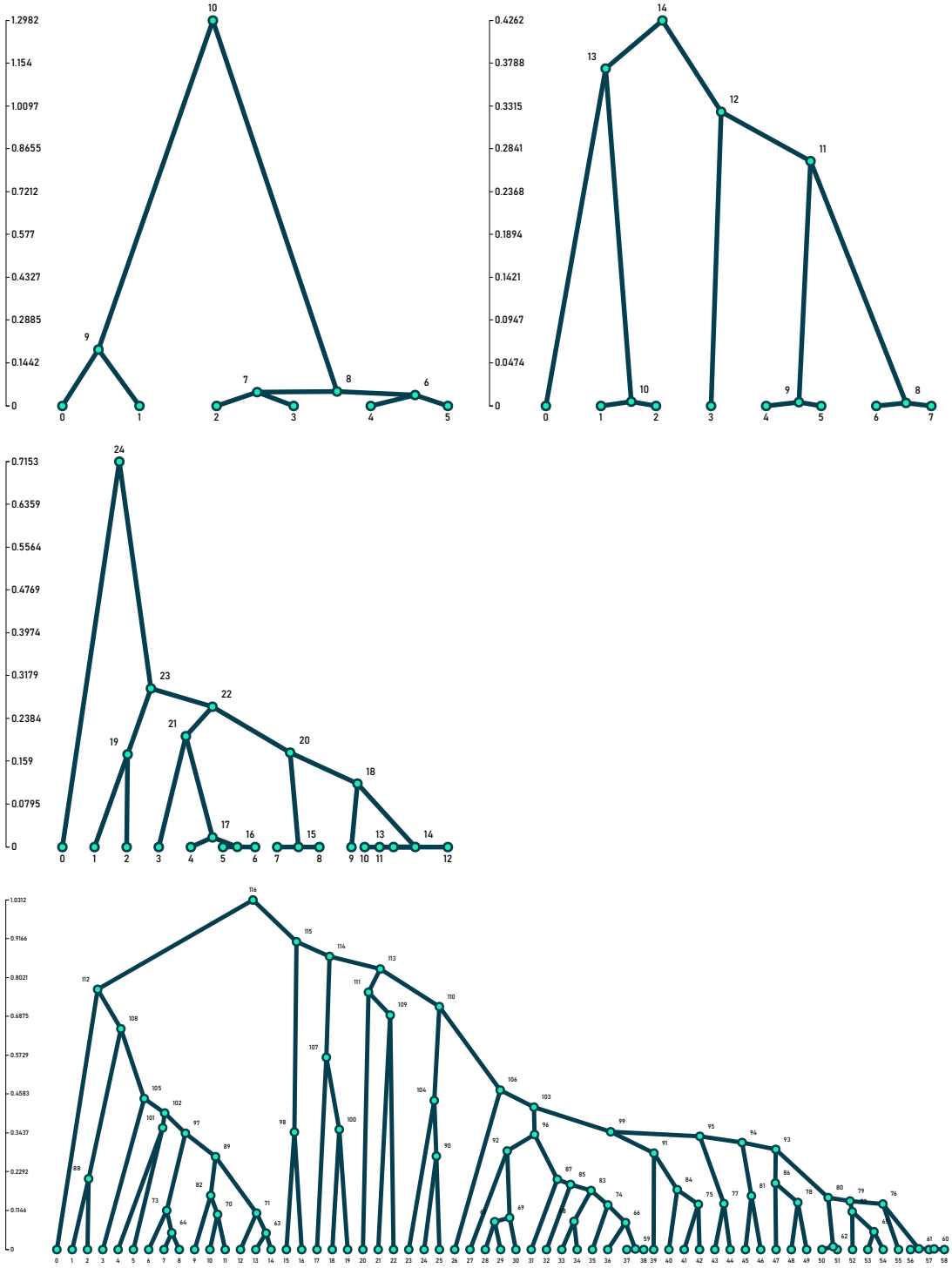


Figure 5.7: Phylogenetic trees inferred by *PanX* and used for parameter estimation. From top to bottom: Bartonella quintana, Staphylococcus argenteus, Clostridium butyricum and Buchnera aphidicola.

5.5 Software and Package Versions

Name	Version	Reference
bintrees	2.2.0	pypi.org/project/bintrees
matplotlib	3.8.0	matplotlib.org
msprime	1.3.1	tskit.dev/software/msprime.html
numpy	1.26.4	numpy.org/
pandas	2.2.0	pandas.pydata.org
python	3.12.1	python.org
scipy	1.12.0	scipy.org
seaborn	0.12.2	seaborn.pydata.org
statsmodels	0.14.1	statsmodels.org/stable
tskit	0.5.6	tskit.dev/software/tskit.html
tskit-arg-visualizer	0.0.1	pypi.org/project/tskit-arg-visualizer

An installable conda environment is available on GitHub:
pangenome-gene-transfer-simulation/blob/main/conda_env.yml.

5.6 Sustainability

Most of the simulations were run on an Intel ® Core ™ i5-13500 processor with a peak power consumption of 154W. Taking into account additional overheads for cooling, idle GPU and other components, the estimated total power consumption is 200W. The remaining simulations were run on an AMD Ryzen ™ Threadripper ™ 3970X 32-core processor with a peak power consumption of 280W. This would give an approximate total power consumption of 350W. Total test and simulation durations were estimated at 350 and 50 hours respectively.

The resulting energy consumption was 87.5 kWh, supplied by a 100% renewable energy supplier.

Bibliography

[Bau+21] Franz Baumdicker et al. 'Efficient ancestry and mutation simulation with msprime 1.0'. In: *Genetics* 220.3 (Dec. 2021). Ed. by S Browning. ISSN: 1943-2631. DOI: 10.1093/genetics/iyab229. URL: http://dx.doi.org/10.1093/genetics/iyab229.

[BHP10] F. Baumdicker, W. R. Hess and P. Pfaffelhuber. 'The diversity of a distributed genome in bacterial populations'. In: *The Annals of Applied Probability* 20.5 (Oct. 2010). DOI: 10.1214/09-aap657. URL: http://dx.doi.org/10.1214/09-AAP657.

[BHP12] Franz Baumdicker, Wolfgang R. Hess and Peter Pfaffelhuber. 'The Infinitely Many Genes Model for the Distributed Genome of Bacteria'. en. In: *Genome Biology and Evolution* 4.4 (2012), pp. 443–456. DOI: 10.1093/gbe/evs016. URL: http://dx.doi.org/10.1093/gbe/evs016.

[BP14] Franz Baumdicker and Peter Pfaffelhuber. 'The infinitely many genes model with horizontal gene transfer'. In: *Electronic Journal of Probability* 19.none (Jan. 2014). DOI: 10.1214/ejp.v19-2642. URL: http://dx.doi.org/10.1214/EJP.v19-2642.

[Bur15] Alita R. Burmeister. 'Horizontal Gene Transfer'. In: *Evolution, Medicine, and Public Health* 2015.1 (2015), pp. 193–194. ISSN: 2050-6201. DOI: 10.1093/emph/eov018. URL: http://dx.doi.org/10.1093/emph/eov018.

[BXH14] Benjamin Buchfink, Chao Xie and Daniel H Huson. 'Fast and sensitive protein alignment using DIAMOND'. In: *Nature Methods* 12.1 (Nov. 2014), pp. 59–60. ISSN: 1548-7105. DOI: 10.1038/nmeth.3176. URL: http://dx.doi.org/10.1038/nmeth.3176.

[Che+21] Zhuoyu Chen et al. 'Prochlorococcus have low global mutation rate and small effective population size'. In: *Nature Ecology and Evolution* 6.2 (Dec. 2021), pp. 183–194. ISSN: 2397-334X. DOI: 10.1038/s41559-021-01591-0. URL: http://dx.doi.org/10.1038/s41559-021-01591-0.

[CPM19] Rebecca A Chong, Hyunjin Park and Nancy A Moran. 'Genome Evolution of the Obligate Endosymbiont Buchnera aphidicola'. In: *Molecular Biology and Evolution* 36.7 (Apr. 2019). Ed. by Deepa Agashe, pp. 1481–1489. ISSN: 1537-1719. DOI: 10.1093/molbev/msz082. URL: http://dx.doi.org/10.1093/molbev/msz082.

[DBN17] Wei Ding, Franz Baumdicker and Richard A Neher. 'panX: pan-genome analysis and exploration'. In: *Nucleic Acids Research* 46.1 (Oct. 2017), e5–e5. ISSN: 1362-4962. DOI: 10.1093/nar/gkx977. URL: http://dx.doi.org/10.1093/nar/gkx977.

[dev24] tskit developers. 'msprime: Simulate genealogical trees and genomic sequence data using population genetic models. Version 1.3.1'. In: (Feb. 2024). URL: https://github.com/tskit-dev/msprime.

[Ehr+10] Garth D. Ehrlich et al. 'The distributed genome hypothesis as a rubric for understanding evolutionin situduring chronic bacterial biofilm infectious processes'. In: *FEMS Immunology and Medical Microbiology* 59.3 (Aug. 2010), pp. 269–279. ISSN: 1574-695X. DOI: 10.1111/j.1574-695x.2010.00704.x. URL: http://dx.doi.org/10.1111/j.1574-695X.2010.00704.x.

[ESF18] Stefan C. Endres, Carl Sandrock and Walter W. Focke. 'A simplicial homology algorithm for Lipschitz optimisation'. In: *Journal of Global Optimization* 72.2 (Mar. 2018), pp. 181–217. ISSN: 1573-2916. DOI: 10.1007/s10898-018-0645-y. URL: http://dx.doi.org/10.1007/s10898-018-0645-y.

[Flo+13] Pedro Flombaum et al. 'Present and future global distributions of the marine Cyanobacteria Prochlorococcus and Synechococcus'. In: *Proceedings of the National Academy of Sciences* 110.24 (May 2013), pp. 9824–9829. ISSN: 1091-6490. DOI: 10.1073/pnas.1307701110. URL: http://dx.doi.org/10.1073/pnas.1307701110.

[Kat02] K. Katoh. 'MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform'. In: *Nucleic Acids Research* 30.14 (July 2002), pp. 3059–3066. ISSN: 1362-4962. DOI: 10.1093/nar/gkf436. URL: http://dx.doi.org/10.1093/nar/gkf436.

[KEM16] Jerome Kelleher, Alison M Etheridge and Gilean McVean. 'Efficient coalescent simulation and genealogical analysis for large sample sizes'. In: *PLoS computational biology* 12.5 (2016), e1004842.

[Kin82] J. F. C. Kingman. 'On the genealogy of large populations'. In: *Journal of Applied Probability* 19.A (1982), pp. 27–43. ISSN: 1475-6072. DOI: 10.2307/3213548. URL: http://dx.doi.org/10.2307/3213548.

[KP08] Patrick J. Keeling and Jeffrey D. Palmer. 'Horizontal gene transfer in eukaryotic evolution'. In: *Nature Reviews Genetics* 9.8 (Aug. 2008), pp. 605–618. ISSN: 1471-0064. DOI: 10.1038/nrg2386. URL: http://dx.doi.org/10.1038/nrg2386.

[LN21] Kirk E. Lohmueller and Rasmus Nielsen. *Human Population Genomics: Introduction to Essential Concepts and Applications*. Springer International Publishing, 2021, pp. 6–11. ISBN: 9783030616465. DOI: 10.1007/978-3-030-61646-5. URL: http://dx.doi.org/10.1007/978-3-030-61646-5.

[Mes16] P.W. Messer. 'Neutral Models of Genetic Drift and Mutation'. In: *Encyclopedia of Evolutionary Biology*. Elsevier, 2016, pp. 119–123. DOI: 10.1016/b978-0-12-800049-6.00031-7. URL: http://dx.doi.org/10.1016/B978-0-12-800049-6.00031-7.

[PDA10] Morgan N. Price, Paramvir S. Dehal and Adam P. Arkin. 'FastTree 2 - Approximately Maximum-Likelihood Trees for Large Alignments'. In: *PLoS ONE* 5.3 (Mar. 2010). Ed. by Art F. Y. Poon, e9490. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0009490. URL: http://dx.doi.org/10.1371/journal.pone.0009490.

[QD19] Maxime Québatte and Christoph Dehio. 'Bartonella gene transfer agent: Evolution, function, and proposed role in host adaptation'. In: *Cellular Microbiology* 21.11 (July 2019). ISSN: 1462-5822. DOI: 10.1111/cmi.13068. URL: http://dx.doi.org/10.1111/cmi.13068.

[Ric+11] Thomas A. Richards et al. 'Horizontal gene transfer facilitated the evolution of plant parasitic mechanisms in the oomycetes'. In: *Proceedings of the National Academy of Sciences* 108.37 (Aug. 2011), pp. 15258–15263. ISSN: 1091-6490. DOI: 10.1073/pnas.1105100108. URL: http://dx.doi.org/10.1073/pnas.1105100108.

[Sch23] Olaf G. Schmidt. *Genetik und Molekularbiologie*. Springer Berlin Heidelberg, 2023. ISBN: 9783662669471. DOI: 10.1007/978-3-662-66947-1. URL: http://dx.doi.org/10.1007/978-3-662-66947-1.

[SHG15] Shannon M. Soucy, Jinling Huang and Johann Peter Gogarten. 'Horizontal gene transfer: building the web of life'. In: *Nature Reviews Genetics* 16.8 (July 2015), pp. 472–482. ISSN: 1471-0064. DOI: 10.1038/nrg3962. URL: http://dx.doi.org/10.1038/nrg3962.

[SP97] Rainer Storn and Kenneth Price. 'Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces'. In: *Journal of Global Optimization* 11.4 (1997), pp. 341–359. ISSN: 0925-5001. DOI: 10.1023/a:1008202821328. URL: http://dx.doi.org/10.1023/A:1008202821328.

[Tet+05] Hervé Tettelin et al. 'Genome analysis of multiple pathogenic isolates of Streptococcus agalactiae: Implications for the microbial "pan-genome"'. In: *Proceedings of the National Academy of Sciences* 102.39

(Sept. 2005), pp. 13950–13955. ISSN: 1091-6490. DOI: 10.1073/pnas.0506758102. URL: http://dx.doi.org/10.1073/pnas.0506758102.

[Ton+23] Gerry Tonkin-Hill et al. 'Robust analysis of prokaryotic pangenome gene gain and loss rates with Panstripe'. en. In: *Genome Research* 33.1 (Jan. 2023), pp. 129–140. DOI: 10.1101/gr.277340.122. URL: http://dx.doi.org/10.1101/gr.277340.122.

[Vir+20] Pauli Virtanen et al. 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum **Unterschrift**