Title

Analysis and Simulation Methods for Artificial Selection Experiments in the Investigation of the Genetic Basis of Complex Traits

Author

Kessner, Darren

Publication Date

2014

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Analysis and Simulation Methods for Artificial Selection Experiments in the Investigation of the Genetic Basis of Complex Traits

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Bioinformatics

by

Darren Kessner

2014

ABSTRACT OF THE DISSERTATION

Analysis and Simulation Methods for Artificial Selection Experiments in the Investigation of the Genetic Basis of Complex Traits

by

Darren Kessner

Doctor of Philosophy in Bioinformatics

University of California, Los Angeles, 2014

Professor John Novembre, Co-chair

Professor Kenneth Lange, Co-chair

One of the fundamental goals of research in modern genetics is to determine the genetic basis for complex traits. One experimental approach to this problem is artificial selection, where individual organisms are selected each generation for extreme values of the trait under study. In these experiments, the investigator identifies putative trait loci based on genetic differentiation in evolved populations. Recently, researchers have combined artificial selection with genome-wide pooled massively parallel sequencing to identify quantitative trait loci. In this dissertation, I present analysis and simulation methods applicable to pooled sequencing and artificial selection experiments.

In Chapter 1, I provide some background on artificial selection, massively parallel sequencing, and the use of simulations in population genetics.

In Chapter 2, I present an expectation-maximization (EM) algorithm for estimating haplotype frequencies in a pooled sample directly from mapped sequence reads, in the case where the possible haplotypes are known. This method is relevant to the analysis of pooled sequencing data from selection experiments, in addition to the calculation of proportions of different species

within a metagenomics sample. The method outperforms existing methods based on single-site allele frequencies, as well as simple approaches using sequence read data. I have implemented the method in a freely available open-source software tool called `harp` (Haplotype Analysis of Reads in Pools). In Appendix A, I present additional analyses to show that the method improves estimates of relative abundances and community diversity at higher taxon levels.

In Chapter 3, I present a new forward-in-time simulator `forqs` (Forward-in-time simulation of Recombination, Quantitative traits, and Selection). `forqs` was designed to investigate haplotype patterns resulting from scenarios where substantial evolutionary change has taken place in a small number of generations due to recombination and/or selection on polygenic quantitative traits. The simulator uses a memory-efficient representation of chromosomes that allows the simulation of whole genomes. In addition, `forqs` explicitly models quantitative traits, and its modular design gives the user great flexibility in specifying trait architectures, selection and demography.

In Chapter 4, I present a new analysis of the power of artificial selection experiments to detect and localize quantitative trait loci (QTLs), using the `forqs` simulator from Chapter 3. I show that modeling loci with constant selection coefficients does not fully capture the dynamics of QTLs under artificial selection. I also show that a substantial portion of the genetic variance of the trait (50–100%) can be explained by detected QTLs in as little as 20 generations of selection, depending on the trait architecture and experimental design. Furthermore, I show that the power to detect and localize QTLs depends crucially on the opportunity for recombination during the experiment. Finally, I show that an increase in power is obtained by leveraging founder haplotype information to obtain allele frequency estimates (using the `harp` method from Chapter 2).

The dissertation of Darren Kessner is approved.

Eleazar Eskin

Janet Sinsheimer

Robert Wayne

Kenneth Lange, Committee Co-chair

John Novembre, Committee Co-chair

University of California, Los Angeles

2014

This dissertation is dedicated to my wife Laura Jane
and my daughters Emma and Isabelle
for all of their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor John Novembre for all of his guidance and support throughout my graduate studies and research. His insights and feedback were invaluable, and he always challenged me to produce work of the highest quality.

I would also like to thank Ken Lange – his inspirational teaching and writing significantly enhanced my graduate experience. I also thank my committee members Eleazar Eskin, Janet Sinsheimer, and Bob Wayne for their support and encouragement. Additionally, I thank my lab colleagues Alex Platt, Charleston Chiang, Eunjung Han, and Diego Ortega Del Vecchyo for creating a great environment for sharing ideas and advice.

Chapter 2 is a version of: Kessner D, Turner T, Novembre J. 2013. Maximum Likelihood Estimation of Frequencies of Known Haplotypes from Pooled Sequence Data. *Molecular Biology & Evolution* 30 (5): 1145-1158. TT provided experimental data, and JN was principal investigator.

Chapter 3 is a version of: Kessner D and Novembre J. 2014. forqs: forward-in-time simulation of recombination, quantitative traits and selection *Bioinformatics* 30 (4): 576–577. JN was principal investigator.

This work was supported by the National Institutes of Health (Training Grant in Genomic Analysis and Interpretation T32 HG002536 for DK, R01 HG007089 for JN), National Science Foundation (EF-0928690 for JN), and UCLA (Dissertation Year Fellowship for DK).

VITA

1989-1993	B.S., Mathematics, UCLA, Los Angeles, California
1993-1995	M.A., Mathematics, Princeton University, Princeton, New Jersey
1997-2001	Software Engineer, Symantec Corporation, Santa Monica, California
2001-2002	Software Engineer, Sony Coporation, Culver City, California
2002-2003	Computer Science Teacher, Oakwood School, North Hollywood, California
2004-2009	Scientific Programmer, Cedars-Sinai Medical Center, Los Angeles, California
2010-2013	Graduate Student Trainee, Genomic Analysis Training Program, UCLA, Los Angeles, California
2013-2014	Dissertation Year Fellowship, UCLA, Los Angeles, California
2013-2014	University Teaching Fellow, UCLA, Los Angeles, California

PUBLICATIONS

Kessner D and Novembre J (2014). forqs: forward-in-time simulation of recombination, quantitative traits and selection *Bioinformatics* 30 (4): 576–577.

Kessner D, Turner T, Novembre J (2013). Maximum Likelihood Estimation of Frequencies of Known Haplotypes from Pooled Sequence Data. *Molecular Biology & Evolution* 30 (5): 1145-1158.

Nelson MR, Wegmann D, Ehm MG, Kessner D, et al. (2012). An Abundance of Rare Functional Variants in 202 Drug Target Genes Sequenced in 14,002 People. *Science* 337:100-104.

Rubio JP, Topp S, Warren L, St Jean PL, Wegmann D, Kessner D, et al. (2012). Deep sequencing of the LRRK2 gene in 14,002 individuals reveals evidence of purifying selection and independent origin of the p.Arg1628Pro mutation in Europe. *Human Mutation* 33:1087-1098.

Chambers M, . . . , Kessner D, et al. (2012). A cross-platform toolkit for mass spectrometry and proteomics. *Nature Biotechnology* 30, 918-920.

Wegmann D, Kessner DE, et al. (2011). Recombination rates in admixed individuals identified by ancestry-based inference. *Nature Genetics* 43:847-853.

Martens L, Chambers M, Sturm M, Kessner D, et al. (2011). mzML – a community standard for mass spectrometry data. *Molecular & Cellular Proteomics* 10:R110.000133.

Luethy R, Kessner DE, et al. (2008). Precursor-Ion Mass Re-Estimation Improves Peptide Identification on Hybrid Instruments. *Journal of Proteome Research* 7:4031–4039.

Kessner D, Chambers M, Burke R, Agus D, Mallick P (2008). ProteoWizard: open source software for rapid proteomics tools development. *Bioinformatics* 24:2534-2536.

CHAPTER 1

Introduction

One of the fundamental goals of modern genetics research is to determine the genetic basis for complex traits. In particular, a major area of ongoing research is the mapping of loci underlying quantitative traits (quantitative trait loci, QTLs). One method for mapping QTLs relies on artificial selection of individuals for extreme values of the trait, which results in population allele frequency changes at loci contributing to the trait. Recent advances in DNA sequencing technology have enabled researchers to easily and inexpensively scan the genome for putative QTLs on the basis of these allele frequency changes. This combination of artificial selection followed by genome-wide pooled sequencing is known as the Evolve and Resequence (E&R) method.

The research I present in this dissertation involves new methods for analyzing and simulating E&R experiments. Chapter 2 is applicable to the analysis of pooled sequence data from E&R experiments, and in Chapters 3 and 4, I present simulation methods I developed and analysis I performed to guide the design of future experiments. In this short chapter, I provide context for this research by presenting some background on the E&R method and the use of simulation methods in population genetics.

1.1 The Evolve and Resequence method for mapping QTLs

1.1.1 Artificial selection and QTL mapping

Artificial selection has long been used by humans to breed plants and animals for desired characteristics, such as increased dairy production in livestock, or greater yield in fruits and grains. Over the past century, artificial selection has also been used extensively as an experimental technique in genetics research. In an artificial selection experiment, researchers subject a population to selection pressure that it would not normally experience in nature. This selection pressure may take the form of, for example, a change in environmental conditions (e.g. temperature or nutrient availability), or direct selection of particular individuals possessing a desired phenotype.

Some of the first artificial selection experiments, performed in the early 1900s, were designed to settle ongoing debates about the nature of selection. In particular, early researchers hoped to answer questions about whether selection on continuous variation was even possible, and how to reconcile this with the Mendelian viewpoint of genes as discrete heritable units (see Falconer (1992) for a history of early selection experiments).

Traditional QTL mapping studies begin with two inbred parental lines that differ in their mean trait values; often these inbred lines are obtained through artificial selection (see Mackay *et al.* (2009) for a review of QTL mapping methods). After creating an F_1 generation by crossing the parental lines, F_1 individuals are mated to create an F_2 mapping population. Individuals in the mapping population have haplotypes that are mosaics of the inbred parental haplotypes. These individuals are measured for the trait, and genotyped at markers that segregate between the parental populations. QTLs are then identified by correlation between trait value and genotype at a locus, where the genotype is inferred by linkage to neighboring markers. The ability to localize a QTL using this method depends crucially on the scale of recombination in the mapping population. For this reason, recombinant inbred lines (RILs) are often used for the mapping population to increase mapping resolution. RILs are derived from the F_2 generation by repeated

sib mating, resulting in the accumulation of more recombination breakpoints between parental haplotypes. This accumulation slows each generation as homozygosity increases in the RIL population, eventually stopping when the RIL is fully inbred. To further increase the amount of recombination in the mapping population, F_2 individuals can be mated randomly to create advanced intercross lines (AILs, Darvasi and Soller (1995)), which can be subsequently inbred.

In the context of mapping QTLs, Genome-Wide Association Studies (GWAS) can be seen to be very similar to traditional QTL mapping (see Korte and Farlow (2013) for a review of GWAS for QTL mapping). In GWAS, the mapping population is usually a natural outbred population, and the markers are single-nucleotide polymorphisms (SNPs). The mapping resolution in outbred populations is determined by historical linkage disequilibrium between QTLs and SNPs. In general, this is much finer than in laboratory populations obtained by crossing inbred lines, leading to better localization of QTLs. However, the power to detect a QTL depends on the phenotypic variance explained by the QTL, which is determined by its effect size and allele frequency. Because of this, GWAS generally have low power to detect QTL variants with small effects or that are rare in the population.

Under artificial selection for extreme values of a trait, allele frequencies at QTLs will increase or decrease due to the selection pressure. While traditional QTL mapping and GWAS both identify QTLs on the basis of cosegregation of genotype and phenotype, an alternative strategy is to identify QTLs by scanning the genome for loci that exhibit large allele frequency changes. This strategy has been implemented using array-based genotyping in a wide variety of organisms, including for example maize (Laurie *et al.*, 2004), yeast (Ehrenreich *et al.*, 2010), fruit flies (Lai *et al.*, 2007; Nuzhdin *et al.*, 2007; Teotonio *et al.*, 2009), chickens (Johansson *et al.*, 2010), and mice (Keightley and Bulfield, 1993).

With the development of massively parallel sequencing technology, allele frequency changes in evolved populations can now be measured by genome-wide pooled sequencing – this is the Evolve and Resequence (E&R) method. In the next sections, I first give an overview of the new

sequencing technology, followed by a more in depth discussion of E&R experiments.

1.1.2 Massively parallel sequencing

The completion of the Human Genome Project and the publishing of the reference human genome (International Human Genome Sequencing Consortium *et al.*, 2001) stimulated development of new and improved methods for sequencing genomes. These technologies, known as next-generation sequencing, have revolutionized the biological sciences by allowing researchers to obtain DNA sequence data easily and affordably.

Several next-generation sequencing machines are commercially available (see Metzker (2010) for an overview). While the specific methods vary by manufacturer, they generally use massively parallel sequencing to obtain millions of short DNA sequence reads from random locations in the genome. As a result, these new sequencing technologies have spurred the development of a wide variety of computational methods to handle the large volume of sequence data, as well as to provide meaningful interpretation of the data in spite of the inherent randomness of the data generation process. For example, when there is a reference genome available for the organism under study, the first analysis step will be to align the raw sequence reads to the reference. Because read alignment is so fundamental, dozens of algorithms and software packages have been developed to perform this step (see Li and Homer (2010) for a review of alignment algorithms).

An additional challenge in the analysis of next generation sequence data comes from the relatively high error rate for individual sequence reads. Consequently, next-generation sequencing machines report base quality scores along with the actual sequence base calls. The base quality score represents an estimate of the probability that the base call is correct. In general, researchers use a combination of two techniques to handle base errors: 1) increase read coverage (the average number of sequence reads per site) to obtain more reliable genotype or allele frequency estimates, and 2) utilize the base quality scores, either filtering out poor quality reads

or incorporating the scores into a likelihood function during downstream analysis.

When population allele frequencies are of primary interest, pooling samples can reduce the cost and labor involved in sample preparation, library construction, and sequencing (Futschik and Schlotterer, 2010). Because sequence reads may originate from any of the individual organisms contributing to the pooled sample, it is not possible to obtain, for example, genotypes of individuals in the pool. However, one can obtain an estimate of the allele frequency of a particular variant in the population by calculating the proportion of reads representing this allele among all reads covering the polymorphic site. While this method is easy to implement, it is subject to error from two sources: random base errors, and stochasticity in the number of reads covering the site.

In Chapter 2, I present a method to obtain local haplotype frequency estimates from pooled sequence data. Because the method incorporates base quality scores and combines information from sequence reads across a genomic region, the estimates are robust to the two primary sources of error in read-count-based allele frequency estimates. In Chapter 4, I show that using the local haplotype frequency estimates to derive allele frequencies leads to greater power to detect and localize loci contributing to a quantitative trait.

1.1.3 E&R experiments

The Evolve and Resequence method combines artificial selection with next-generation genome-wide pooled sequencing to study the genetic basis of complex traits. At the end of the selection experiment, the selected populations are pooled and sequenced to obtain genome-wide allele frequency estimates. The genome is then scanned for variants whose allele frequencies have changed substantially over the course of the experiment. A site exhibiting a large allele frequency difference indicates the presence of a nearby locus contributing to the trait (Figure 1.1).

While E&R experiments can be performed on any organism (for example Parts *et al.* (2011) studied heat tolerance in yeast), the vast majority of E&R studies have used the fruit fly

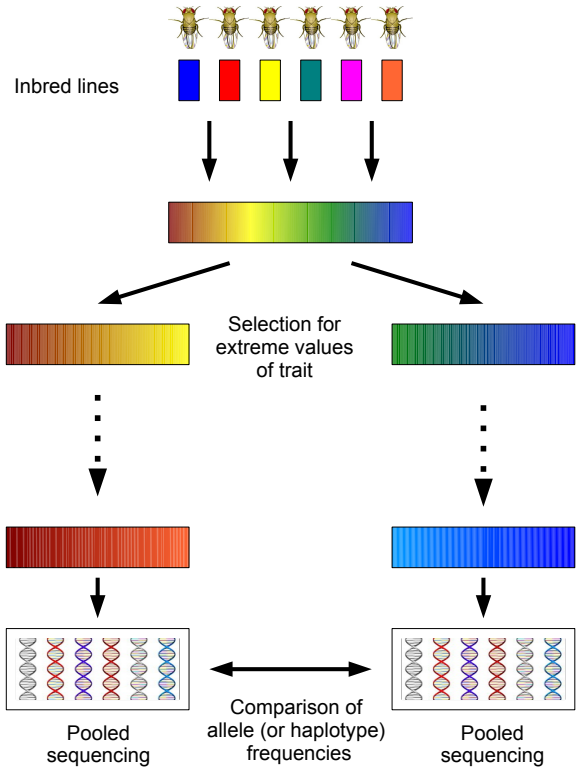

Figure 1.1: **Evolve and resequence experiment.** After initial neutral mixing of founders, individuals with extreme values of the trait are selected to create the next generation. After several generations of selection, populations are sequenced and analyzed for allele frequency differences.

Drosophila melanogaster. *D. melanogaster* is the species of choice because populations are easy to maintain in the laboratory, generation times are short ($\sim$ 2 weeks), and also because several techniques have been developed for fine mapping of candidate QTLs (see Mackay (2001) for a review).

Several groups have successfully identified QTLs for various traits in *Drosophila* via the E&R method. Burke *et al.* (2010) performed whole-genome sequencing of populations that had been selected for accelerated development for over 600 generations and found a large number

of genomic regions that had differentiated between selected and control populations. Turner *et al.* (2011) divergently selected two replicate populations each for large and small body size for over 100 generations, finding strong evidence for QTLs contributing to the trait. To identify genes that contribute to hypoxia tolerance, Zhou *et al.* (2011) exposed populations of flies to hypoxic conditions for over 200 generations, finding several candidate genes that they also functionally validated. Remolina *et al.* (2012) selected populations for 50 generations to find genes related to life history traits (life span and fecundity). Orozco-terWengel *et al.* (2012) sequenced populations at three time points over 37 generations as they adapted to a novel laboratory environment, finding substantial variation in the allele frequency trajectories of selected variants. In a shorter experiment (14 generations of selection), Turner and Miller (2012) selected male flies divergently based on the interpulse interval of the male courtship song, finding significant differentiation at variants across the genome, albeit with poor mapping resolution.

Although information about the individuals comprising the final population is lost when they are pooled together, investigators may have additional information about the founders of the population. For example, Turner and Miller (2012) used fully sequenced inbred lines from the Drosophila Genetic Reference Panel (DGRP) (Mackay *et al.*, 2012) to create the founding population for the selection experiment. In such an experiment, individual haplotypes in the evolved populations will be, apart from the rare occurrence of *de novo* mutations, mosaics of haplotypes from the founding population, whose sequences are known.

In Chapter 2 (and Appendix A), I show how information about the founder haplotypes can be leveraged to obtain local haplotype frequency estimates from pooled sequence data. While my primary motivation was the analysis of data from E&R experiments, this method is also applicable to metagenomics studies, where it can be used to obtain relative abundance estimates of known taxa in a naturally pooled sample. In Chapter 4, I report the results of a power analysis I performed by computer simulation of E&R experiments. Because existing simulation software was not adequate for this analysis, I designed and implemented a new forward simulator which

I describe in Chapter 3.

1.2 Simulations in population genetics

Computer simulations play a substantial role in population genetics research. Simulations are used to verify analytical results, validate statistical methods, perform statistical inference, and generate predictions from analytically intractable mathematical models. *Forward-in-time* (e.g. Wright-Fisher) simulations follow individuals (or individual chromosomes) in a population through multiple generations, while *backward-in-time* (e.g. coalescent) simulations start from the present and trace lineages back in time as they coalesce into a single lineage.

Backward simulations (e.g. `ms` (Hudson, 2002), `MaCS` (Chen *et al.*, 2009), `fastsimcoal` and `fastsimcoal2` (Excoffier and Foll, 2011; Excoffier *et al.*, 2013)) are very efficient for simulating neutral sequence data, due to the fact that they only trace extant lineages. In contrast, forward simulations track all individuals in a population, even those who leave no present-day descendants. The efficiency of backward simulations is very useful when modeling the effects of demographic history (e.g. population size changes, population merges/splits, migrations) on the genomes of large populations over long timescales. However, while it is possible to simulate certain selection scenarios within the coalescent framework (Hudson and Kaplan, 1988; Ewing and Hermisson, 2010), one must turn to forward-in-time simulations to model selection in a flexible way.

Forward simulations offer the ability to model phenomena such as selection, mating systems, and complex phenotypes. In Chapter 3, I discuss a new forward simulator that I designed and implemented, called `forqs` (Forward-in-time simulation of Recombination, Quantitative Traits, and Selection). The software employs a memory-efficient model allowing the fast simulation of whole genomes, while also offering flexible specification of quantitative traits and complex selection scenarios. In the following, I discuss the basic principles underlying forward

simulations, beginning with the Wright-Fisher model for completeness.

1.2.1 Wright-Fisher model

The Wright-Fisher model is perhaps the most basic model used in population genetics. In this model, we consider a single population of N haploid individuals, distinguished only by the allele they carry at a single polymorphic locus. We assume that there are only two possible alleles at this locus, which we label 0 and 1. We further assume that generations are non-overlapping. To create a new individual in the next generation, a parent is chosen at random from the current generation, and the new individual receives a copy of the allele carried by the parent. Note that the parent is not removed, so that individuals in the current generation may have multiple offspring in the next generation. We can extend this model to diploid individuals by specifying each individual by the two alleles they carry. In the diploid case, two parents are chosen randomly from the current population, and from each parent one allele is chosen randomly to be copied to the child. In this case, there are $2N$ total copies of the locus in the population. We restrict our attention to the diploid case from now on.

If we let p be the frequency of allele 1 in the population in the current generation, then the count C of allele 1 in the next generation will be binomially distributed: $C \sim Binom(p, 2N)$. The allele frequency $P = C/2N$ in the next generation is thus a random variable with expectation $E(P) = p$ and variance $Var(p) = p(1-p)/2N$. From this it can be seen that the magnitude of genetic drift (the random fluctuation in allele frequency from one generation to the next) depends on $1/2N$. As a result, larger populations will experience smaller changes in allele frequency each generation than smaller populations. We observe that if $p = 0$ (allele 1 is lost) or if $p = 1$ (allele 1 is fixed) in any generation, then that allele remains lost or fixed in subsequent generations. We additionally observe that because there is a chance of loss or fixation of the allele each generation, the allele will be lost or fixed eventually.

To simulate the Wright-Fisher model, we can simply represent the population at a given

time by the single number p, the allele frequency. We then generate values of p in subsequent generations by binomial sampling as described above.

1.2.2 Extensions to the model

We can add mutation to the model by specifying a mutation rate μ at which the allele copied from a parent mutates to the other allele before being transmitted to the child. Alternatively, two mutation rates can be specified, for forward ($0 \rightarrow 1$) and backward ($1 \rightarrow 0$) mutation.

We can also add selection to the model, by assigning a fitness to each genotype. In this case, parents are chosen randomly with probability proportional to their fitness. We can represent the fitness corresponding to a particular genotype as $1 + s$, where the parameter s, the selection coefficient, represents the proportional fitness advantage gained by individuals with that genotype.

To extend the model to accommodate multiple loci, we first make the observation that each individual will carry two of 2^L possible haplotypes, where L is the number of loci. Additionally, we could allow more than two alleles at each locus. For example, if a locus is a single site with any of 4 nucleotides possible, there would be 4^L possible haplotypes.

To simulate the multiple locus case, we can proceed in two ways: 1) store the frequency of each possible haplotype for each generation, or 2) store two arrays of length L for each individual that represent the individual's two chromosomes/haplotypes. Method 1 would require memory usage proportional to the number of possible haplotypes, which is exponential in the number of loci L. Method 2 would require memory usage proportional to LN, which is linear in both the number of loci and the population size. As the number of loci increases, the number of possible haplotypes quickly grows to exceed the number of haplotypes actually present in the population (maximum 2N), so Method 2 is preferable for modeling large genomic regions.

Modeling multiple loci opens up the possibility to simulate recombination. To do this, we can give each parent the ability to transmit a mosaic haplotype resulting from recombination

between the parent's two chromosomes. One way to do this is to specify a recombination rate ρ for the population. After a parent is chosen, we draw a random number of crossover positions according to a Poisson distribution with mean ρ, and draw the positions uniformly randomly from the loci under consideration. The recombined haplotype is then constructed by alternately copying the allele at each locus from one parental haplotype or the other, switching at crossover positions.

We also note that one can simulate multiple populations, as well as population size changes and migrations, in a straightforward manner. With these extensions, a great variety of demographic scenarios can be simulated.

1.2.3 Fitness functions

To model selection in the multiple locus case, we need to specify a fitness function that depends on the multilocus genotype of the individual. One typical way to do this is to assign a fitness to each genotype at each locus, parametrized by a selection coefficient (and possibly a dominance parameter). A multilocus fitness function is used to calculate an individual's fitness based on the single site fitnesses. Many forward simulators take this approach (e.g. `ForwSim` (Padhukasahasram *et al.*, 2008) `Fregene` (Chadeau-Hyam *et al.*, 2008) `GENOMEPOP` (Carvajal-Rodriguez, 2008) `SFS_CODE` (Hernandez, 2008) `TreesimJ` (O'Fallon, 2010), `SLiM` (Messer, 2013)). One common simplifying assumption is that the loci have independent effects, which implies a multiplicative fitness function, where an individual's fitness is the product of the single-site fitnesses.

Alternatively, we can specify a more complicated fitness function. For example, to model selection on a quantitative trait, we must first decide how to calculate a trait value for each individual. To do this, we must specify 1) how various loci contribute to the trait, and 2) how the environment affects the trait. One simple way to do this is to assume independent additive effects, where each of several loci has an effect size, together with a random Gaussian environmental effect. More complicated models could include dominance or epistatic effects. In

addition to specifying the quantitative trait, we must specify how fitness depends on an individual's trait value (or possibly values of multiple traits). For example, an individual's fitness could decrease with distance from some optimum value. Another possibility is truncation selection, where only individuals whose trait value is above (or below) some threshold are able to have offspring in the next generation.

In addition to my simulation software `forqs`, a few other forward simulators include support for quantitative traits (e.g. `ForSim` (Lambert *et al.*, 2008), `quantiNemo` (Neuenschwander *et al.*, 2008), `simuPOP` (Peng and Kimmel, 2005)). However, these existing simulators each have one or more limitations that make them inadequate for the simulation study that I present in Chapter 4. These limitations include: lack of flexible selection/fitness models, requirement of long initial burn-in to reach mutation-drift equilibrium, lack of haplotype information, and inability to simulate whole genomes due to memory limitations.

1.2.4 Computational limitations

We now restrict our attention to the case where we are simulating a single contiguous region of length L (i.e. each locus is a single site), with 4 possible nucleotide bases at each position. By using 2 bits to represent the 4 possible bases, we can store the bases for 4 positions in a single byte (1 byte = 8 bits), for a total of $LN/2$ bytes of memory to store the whole population (for a single generation). Here we can see that we have a tradeoff between the number of individuals and the length of the genomic region simulated. For example, 16GB of memory will allow storage of 1 million individuals with 32 Kb of sequence per individual, or 10000 individuals with 3.2 Mb of sequence per individual. Note that in this latter case, our regions are still more than an order of magnitude smaller than the smallest human chromosome.

In typical scenarios, the majority of sites in a region will be monomorphic in a population. Because of this, one can make more efficient use of memory by storing only the mutations carried by individuals. This strategy is employed by the forward simulator SLiM (Messer, 2013),

for example.

Further efficiency gains in memory usage can be obtained in scenarios where there are a limited number of loci with fitness effects. For example, consider an artificial selection experiment, where selection on a quantitative trait is performed for a small number of generations. In this case, selection acts primarily on standing variation, i.e. only on QTLs that are already polymorphic in the population. In such scenarios, evolution in the population proceeds primarily through recombination, and *de novo* mutations are not expected to play a large role. Because of this, we can store the haplotypes of the founder individuals only, and then represent each descendent individual haplotype as a list of founder haplotype chunks.

In Chapter 3, I describe how I have implemented this approach in the software `forqs`. By employing this strategy, `forqs` is able to simulate whole genomes efficiently (e.g. 23 chromosome pairs, 100 Mb each, for human populations).

1.2.5 Applications of simulations to evaluate E&R studies

Two previous simulation studies have used forward simulations to assess the power of Evolve & Resequence experiments to detect and localize quantitative trait loci (QTLs) (Kofler and Schlotterer, 2014; Baldwin-Brown *et al.*, 2014). Both of these studies used forward simulators where selection at a locus is modeled with a single parameter, the selection coefficient.

In Chapter 4, I describe a simulation study I performed using the `forqs` software. In my simulations, I explicitly model the genome-wide architecture of quantitative traits, with truncation selection based on individual trait values. By explicitly modeling selection on a quantitative trait I am able to parametrize the simulated experiments and report results using the standard quantitative genetics concepts of effect size, genetic variance, and heritability. I show that population genetics simulations based on loci with constant selection coefficients do not fully capture the dynamics of QTLs contributing to a trait under artificial selection. Indeed, the behavior of a QTL under artificial selection is dependent both on the experimental design (proportion of indi-

viduals selected each generation) and on the trait architecture (effect size, and linkage to other QTLs). While the selection coefficient conflates these parameters into a single number, my simulation framework allows these parameters to be separated and investigated independently. In addition, I show the important role that recombination plays in the ability to detect and localize QTLs with an E & R experiment.

1.3 Roadmap

In the following chapters, I present the methods I have developed for the analysis of Evolve and Resequence experiments.

In Chapter 2, I present a method for estimating local haplotype frequencies from pooled sequence data, where the founder haplotypes are known. While the primary motivation of this work was for the analysis of data from E&R experiments, the method is also applicable to the analysis of sequence data from naturally pooled metagenomics samples. Some further analysis of this application is presented in Appendix A.

In Chapter 3, I present the forward simulation software `forqs` that I designed and implemented for the simulation of whole genomes with explicit modeling of quantitative traits and selection. In addition to the simulation of artificial selection experiments, many other selection scenarios can be simulated, such as selection toward a trait value optimum, or fitness depending on multiple quantitative traits.

In Chapter 4, I present an analysis of the power of E&R experiments to detect and localize QTLs. This analysis relies on `forqs` to explicitly model various quantitative trait architectures and experimental designs. Because the results are reported using familiar concepts from quantitative genetics, this analysis will be useful for guiding the design of future E&R experiments.

CHAPTER 2

Maximum Likelihood Estimation of Frequencies of Known Haplotypes from Pooled Sequence Data

2.1 Abstract

DNA samples are often pooled, either by experimental design, or because the sample itself is a mixture. For example, when population allele frequencies are of primary interest, individual samples may be pooled together to lower the cost of sequencing. Alternatively, the sample itself may be a mixture of multiple species or strains (e.g. bacterial species comprising a microbiome, or pathogen strains in a blood sample). We present an expectation-maximization (EM) algorithm for estimating haplotype frequencies in a pooled sample directly from mapped sequence reads, in the case where the possible haplotypes are known. This method is relevant to the analysis of pooled sequencing data from selection experiments, as well as the calculation of proportions of different species within a metagenomics sample. Our method outperforms existing methods based on single-site allele frequencies, as well as simple approaches using sequence read data. We have implemented the method in a freely available open-source software tool.

2.2 Introduction

Pooled sequencing is a common experimental method in which DNA samples from multiple individuals are sequenced together. In some contexts, the pooling of individual samples is per-

formed by the researcher; in others, the sample itself is a mixture of multiple individuals. When population allele frequencies are of primary interest, pooled sequencing approaches can reduce the cost and labor involved in sample preparation, library construction, and sequencing (Futschik and Schlotterer, 2010; Cutler and Jensen, 2010; Kofler *et al.*, 2011; Orozco-terWengel *et al.*, 2012; Huang *et al.*, 2012).

For example, in experimental evolution studies, populations are selected for extreme values of a trait over several generations, followed by pooled sequencing to calculate allele frequencies at polymorphic sites across the genome (Nuzhdin *et al.*, 2007; Burke *et al.*, 2010; Earley and Jones, 2011; Turner *et al.*, 2011; Zhou *et al.*, 2011). Typically, differences in single-site allele frequencies between an experimental population and a control population (or between two experimental populations selected in opposite directions) are used to identify regions of the genome that may have undergone selection during the course of the experiment, and thus contribute to the trait of interest. However, localizing such regions would be improved if haplotype frequencies were more easily estimated from pooled data, as many of the most powerful tests for selection rely on haplotype information (Voight *et al.*, 2006; Sabeti *et al.*, 2007).

In certain cases, haplotype frequency estimation may be more feasible than others, such as when the investigator has prior knowledge about the founders of the pooled sample. For example, Turner and Miller (2012) used inbred lines from the Drosophila Genetic Reference Panel (DGRP) (Mackay *et al.*, 2012) to create the founding population for the selection experiment. In such an experiment, individual haplotypes in the evolved populations will be, apart from *de novo* mutations, mosaics of haplotypes from the founding population, whose sequences are known. This structure should make it simpler to estimate haplotype frequencies, and in turn detect regions harboring adaptive variation, by searching for haplotypes that have increased in frequency locally during the experiment.

In many other contexts, biological samples are naturally pooled, and the researcher is interested in the relative proportions of various species or strains within the sample. For example,

malaria researchers interested in drug resistance and vaccine efficacy testing have developed several laboratory and computational techniques for determining the proportions of different malaria parasite strains in blood samples (Cheesman *et al.*, 2003; Hunt *et al.*, 2005; Takala *et al.*, 2006; Li *et al.*, 2007; Hastings and Smith, 2008; Hastings *et al.*, 2010). In metagenomics studies, one major interest is the relative abundance of different microbial strains and species in pooled samples from different tissues/habitats (Ley *et al.*, 2006; Human Microbiome Project Consortium, 2012). Short sequence reads from 16S rRNA are commonly used to estimate these frequencies by classifying reads by taxon and counting the number of reads in each category (Mizrahi-Man *et al.*, 2013). In these examples, canonical haplotypes (e.g. 16S reference sequence) of many species of interest are known, and accurate estimates of relative frequencies of the known species are of great importance.

Indirect estimation of haplotype frequencies from unphased genotype data has a long history (see Niu (2004) for a review of these methods). Several approaches for estimating haplotype frequencies from pools containing multiple individuals have focused on the use of single-nucleotide polymorphism (SNP) allele frequencies obtained by array-based genotyping (Pe'er and Beckmann, 2003; Ito *et al.*, 2003; Wang *et al.*, 2003; Yang *et al.*, 2003; Kirkpatrick *et al.*, 2007; Zhang *et al.*, 2008; Kuk *et al.*, 2009). Some examples of this class of methods have incorporated prior knowledge about haplotypes in the sample into the estimation (Gasbarra *et al.*, 2009; Pirinen, 2009). Most recently, Long *et al.* (2011) have proposed a method for estimating haplotype frequencies from SNP allele-frequency data obtained by pooled sequencing, using a regression-based approach with known haplotypes.

Pooled sequence data provide two important sources of information beyond single-site allele frequencies: haplotype information from sequence reads that span multiple variant sites, and base quality scores, which give error probability estimates for each base call. Here we introduce a method to use this additional information to estimate haplotype frequencies from pooled sequence data, in the case where the constituent haplotypes are known. This method

uses a probability model that naturally incorporates uncertainty in the reads by using the base quality scores reported with the sequence data. The method obtains a maximum likelihood estimate of the haplotype frequencies in the sample via an expectation-maximization (EM) algorithm (Dempster *et al.*, 1977). We present results from realistic simulated data to show that the method outperforms allele-frequency based methods, as well as simple approaches that use sequence reads. The use of a fixed list of known haplotypes allows the algorithm to use data from much larger genomic regions than algorithms that enumerate all possible haplotypes in a region, which leads to much improved haplotype frequency estimates. We also explore the effects of unknown haplotypes being included in the mixture, and specify conditions affecting the accuracy of the estimation. We have implemented the method in an open-source software tool `harp` (see authors' websites for software link).

2.3 New Approaches

We assume that there are H haplotypes represented in the pool, and that the sequence reads have been generated randomly according to the frequencies of the haplotypes. Informally, we use haplotype information contained in an individual read to probabilistically assign that read to one or more of the known haplotypes (Figure 2.1), and then use the probabilistic haplotype assignments to estimate the haplotype frequencies.

2.3.1 Probability Model

Let $f = (f_1, \ldots, f_H)$ be the frequencies of the H haplotypes in the genomic region of interest. We can think of N sequence reads $r = (r_1, \ldots, r_N)$ as being independently generated as follows. To generate read r_j:

- choose the haplotype η_j to copy from:
  $\eta_j \sim$ Discrete(f)

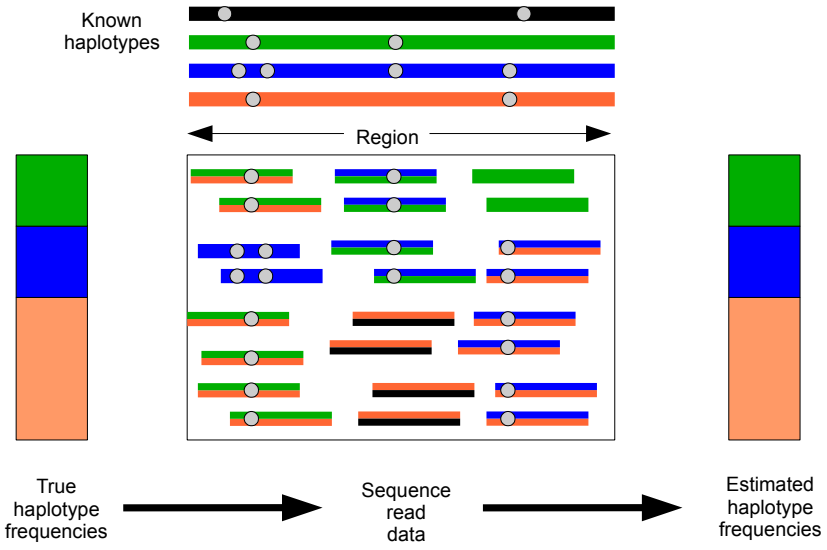

Figure 2.1: **Haplotype information from individual reads can be combined across a genomic region to obtain haplotype frequency estimates.** In this cartoon, there are 4 known haplotypes (black, green, blue, orange), with sequence data coming from a pool containing 25% green, 25% blue, and 50% orange haplotypes. Each read is probabilistically assigned to the known haplotypes. Some reads can be assigned with great certainty, e.g the reads coming from the blue haplotype that cover two neighboring variant sites. Other reads (represented by two colors) are assigned with less certainty.

- choose a starting position uniformly at random in the genomic region, and copy read r_j from haplotype η_j starting at the chosen position
- draw base quality scores for the read from a fixed distribution (which can be determined empirically)
- introduce errors in the sequence read, with the probability of error in a base call given by the base quality score at that position

In practice, haplotypes may not be perfectly known, or there may be segregating variation within the strain represented by a particular haplotype. In such cases, International Union of Pure and Applied Chemistry (IUPAC) ambiguous base codes (e.g. R for purine, Y for pyrimidine, N for any, etc.) may be used in place of the standard bases (A, C, G, T) to indicate the uncertainty. We incorporate these cases into our probability model by assuming the true base at each segregating site is sampled from a discrete distribution with probabilities determined by the allele frequencies at that site within the strain (which may be known a priori or assumed to be uniform).

2.3.2 Haplotype Likelihood

Calculating the likelihood of a set of haplotype frequencies given read data under this model can be carried out as follows. Let L_j be the length of the j^{th} read r_j, let $(r_j[1], \ldots, r_j[L_j])$ be the base calls, and let $q_j = (q_j[1], \ldots, q_j[L_j])$ be the base quality scores. Also, let $(\eta_j[1], \ldots, \eta_j[L_j])$ be the corresponding bases of haplotype η_j. At read position i, $q_j[i]$ is the probability of sequencing error at that position: $q_j[i] = P(r_j[i] \neq \eta_j[i])$. Note that for paired-end data, r_j represents a read pair coming from a single haplotype, and that the positions within the read may not be contiguous.

We have $P(\eta_j, r_j|f, q_j) = P(\eta_j|f)P(r_j|\eta_j, q_j)$. The first term $P(\eta_j|f)$ is given by the discrete distribution with probabilities f, and the second term $P(r_j|\eta_j, q_j)$, the *haplotype likelihood*, can be calculated from the base quality scores, as follows.

First, we assume that sequencing errors within a single read are independent of each other:

$$P(r_j \mid \eta_j, q_j) = \prod_{i=1}^{L_j} P(\, r_j[i] \mid \eta_j[i], q_j[i])$$

Next, we need to specify how to calculate the terms in the above product, i.e. the probability of an observed base, given the true base and the base quality at that position. For simplicity, we

assume that each of the 3 incorrect bases will be observed with equal probability:

$$P(\, r_j[i] \mid \eta_j[i], q_j[i]) = \begin{cases} 1 - q_j[i] & \text{if } r_j[i] = \eta_j[i] \\ q_j[i]/3 & \text{if } r_j[i] \neq \eta_j[i] \end{cases}$$

More generally, we note that we can use a base error matrix (parametrized by base quality score) to allow for unequal probabilities, and that these probabilities can be estimated from the data by considering the monomorphic sites in the sample.

Note that if position i is a segregating site in the strain represented by haplotype η_j, the likelihood is calculated by summing over the possible bases:

$$P(\, r_j[i] \mid \eta_j[i], q_j[i]) = \\ \textstyle\sum_{b \in \{A,C,G,T\}} P(\, r_j[i] \mid \eta_j[i] = b,\, q_j[i])\; P(\eta_j[i] = b)$$

where $P(\eta_j[i] = b)$ is the frequency of base b at that site within the strain. For sites where the possible bases are known, but not the allele frequencies, we set the allele frequencies to be equal, e.g. .5 for biallelic sites, and .25 for sites with no information.

For clarity, we suppress the dependence on the base quality scores in what follows.

2.3.3 Simple Approaches

We explored two simple approaches for estimating haplotype frequencies. The first method is a simple string match algorithm, where sequence reads are fractionally assigned (with equal weight) to haplotypes with which they are identical up to a specified maximum number of mismatches. For example, a read that matches two haplotypes is assigned .5 to each. The fractional assignments are then summed, to obtain counts for each haplotype, and dividing by the number of reads gives the haplotype frequency estimate.

The second method, which we call a *soft* string match, uses the probability model described above to calculate the vector of haplotype likelihoods l_j for each read r_j. Thus, the soft string

match makes use of the base quality scores from the reads. The haplotype likelihood vector l_j is normalized so that the components sum to 1, which we take to be our probabilistic haplotype assignment. As with the fractional assignments above, the probabilistic assignments are averaged to obtain the haplotype frequency estimate.

2.3.4 EM Algorithm

In addition to the simple approaches, we developed a full likelihood approach to obtain maximum likelihood estimates of the haplotype frequencies under the probability model described above.

We assume that our reads are generated independently, so our complete data likelihood is:

$$L(f \mid \eta, r) \;=\; P(\eta, r \mid f) \;=\; \prod_{j=1}^{N} P(\eta_j, r_j | f)$$

We observe the reads r, but treat the haplotype assignments η as missing data, so we are interested in the marginal likelihood,

$$L(f \mid r) \;=\; P(r \mid f) \;=\; \sum_{\eta} P(\eta, r | f)$$

which we maximize by iteratively calculating haplotype frequency estimates via the EM algorithm: $f^{(0)}, f^{(1)}, \ldots$

First we describe the iteration step of the algorithm; we assume we have $f^{(i)}$ and show how to obtain $f^{(i+1)}$. In Material and Methods, we show that this is the formal EM algorithm of Dempster *et al.* (1977).

We let $l_{j,h} = P(r_j | \eta_j = h)$, and let $l_j = (l_{j,1}, \ldots, l_{j,H})$ be the vector of haplotype likelihoods for read j. Note that for a given sequence read r_j, the haplotype likelihood vector l_j is determined by the variant sites covered by the read, up to a proportionality constant. Also note that the haplotype likelihood vectors can be calculated once and cached, before the actual EM iteration.

Given $f^{(i)}$, we define $p_j = (p_{j,1}, \ldots, p_{j,H})$ to be the haplotype posterior vector for read j, where

$$p_{j,h} = P(\eta_j = h | r_j, f^{(i)})$$

Intuitively, p_j is a probabilistic haplotype assignment of read r_j, with each component $p_{j,h}$ representing the probability that the read came from haplotype h (given our current haplotype frequency estimate $f^{(i)}$). Note that:

$$\begin{aligned} P(\eta_j = h | r_j, f^{(i)}) &\propto P(r_j | \eta_j = h)\, P(\eta_j = h | f^{(i)}) \\ &= l_{j,h}\, f_h^{(i)} \end{aligned}$$

so p_j can be obtained by taking the component-wise product $l_j \circ f^{(i)}$, and normalizing so that the vector components sum to 1. As a special case, if $f^{(0)}$ is uniform, then in the first iteration, p_j is just l_j normalized.

Our updated estimate $f^{(i+1)}$ is given by the average of the haplotype posterior vectors:

$$f^{(i+1)} = \frac{\sum_j p_j}{N}$$

Finally, we must specify how to choose our initial haplotype frequency estimate $f^{(0)}$, as well as convergence criteria for the iteration. For our first initial estimate we use the uniform distribution $f_h^{(0)} = 1/H$. We also use additional random initial estimates drawn from a symmetric Dirichlet distribution to start multiple runs of the algorithm, since there is a possibility that the EM algorithm will climb to a non-global local maximum on the likelihood surface. For the termination condition, we specify a threshold ϵ, and halt the iteration when the squared distance between estimates falls below the threshold: $|f^{(i+1)} - f^{(i)}|^2 < \epsilon$. In practice, we found a value of $\epsilon = 10^{-8}$ to work well and this value is used in the results presented below.

2.3.5 Base Quality Score Recalibration

We observed inconsistencies between the reported base quality scores in our experimental data sets and empirical error rates based on sequence reads covering monomorphic sites in the known

haplotypes (see Results), which motivated the development of a recalibration method to correct for these inconsistencies.

Illumina base quality scores have different interpretations, depending on the Illumina version. In our experimental data set, corresponding to Illumina versions 1.5 - 1.7, the scores range from 2 to 40, with the score q representing an error probability given by the Phred scale:

$$P(\text{error}) = 10^{-q/10}$$

For example, a base quality score of 20 gives an error probability of 1/100. The special score of 2 indicates that the base should not be used in downstream analysis.

To recalibrate, we examine monomorphic sites to calculate an observed error rate $P_{obs}(\text{error})(q)$ for each possible base quality score q. These observed error rates can then be used directly in the haplotype likelihood calculation in place of the Phred scale error rates, or to create a new BAM file with recalibrated base quality scores.

2.3.6 Haplotype Likelihood Filtering

The EM algorithm described above relies on the assumption that we know the sequences of the haplotypes found in the pool and that the pool has no contamination from unknown species. While investigating the effects of unknown haplotypes species in the pool, we found that in the case where the unknown is sufficiently unrelated to the known haplotypes (known species, in the case of 16S sequences), reads from the unknown can be filtered out on the basis of the haplotype likelihoods.

For each sequence read, the maximum haplotype likelihood will usually be attained by the haplotype from which the read was derived. Building on this, we can calculate the distribution of the maximum haplotype likelihood of the sequence reads under the assumption that the pool contains only known haplotypes, based on the empirical base quality score distribution of the data (see Material and Methods for details). Using this "null" distribution, we can filter out

reads whose maximum haplotype likelihood falls outside a specified range (Figure 2.8A). In our simulations, we obtained good results by filtering out reads whose maximum haplotype likelihood was less than 2 standard deviations below the mean of this distribution (Figure 2.8C).

2.4 Results

2.4.1 Comparison With Existing Allele-Frequency Based and Simple Sequence Based Methods

We first evaluated the performance of the EM algorithm in comparison to single-site allele-frequency based methods and the simple sequence-based methods discussed above (see New Approaches). To represent the allele-frequency based methods, we chose `hippo`, which is a freely available program that has been shown to outperform other allele-frequency based methods for estimating haplotype frequencies (Pirinen, 2009). One property of this class of methods is that all possible haplotypes in the region are considered during the estimation. This results in an exponential growth in the number of haplotypes (and thus memory usage and algorithm running time) as the region width increases. To improve performance, the `hippo` method allows one to specify known haplotypes, which we do here. We found it difficult to obtain results on our simulated data for regions larger than about 2kb (though this distance scale is driven largely by the relatively high Drosophila-specific levels of diversity we simulated here).

In this comparison, we simulated data from a pool of 20 haplotypes with 100bp paired-end sequence reads and 200x pooled coverage, with 100 replicates each from genomic regions ranging in size from 500bp to 50kb.

We found that the simple methods using sequence reads (string match and soft string match) outperformed the method based on single-site allele frequencies, and that the EM algorithm performed vastly better than all of the other methods (Figure 2.2). The soft stringmatch method

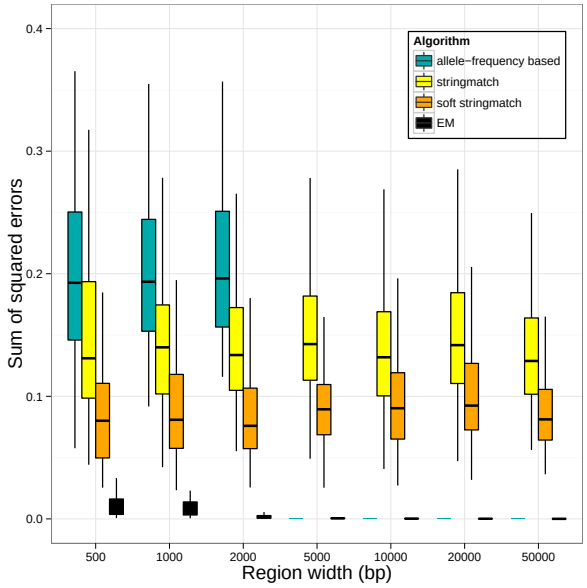

Figure 2.2: **Comparison of the EM algorithm to known allele-frequency based and simple sequence-based methods.** Each algorithm was run on simulated pooled 100bp paired-end sequence data from 20 haplotypes at 200x coverage, with 100 replicates for each region width.

showed a distinct improvement over the stringmatch method, due to the incorporation of information from the base quality scores. We also note that the EM algorithm performed the estimation using 162 reference haplotypes, and accurately reported zero frequencies for the haplotypes not present in the pool.

The EM algorithm's increased performance can be attributed to the sharing of information across all of the reads in the genomic region. In contrast to the other methods, the EM algorithm's performance improves as the width of the region increases. This improvement comes from the fact that more variant sites are available to distinguish between haplotypes, in addition to the increased amount of data on which to base the inference.

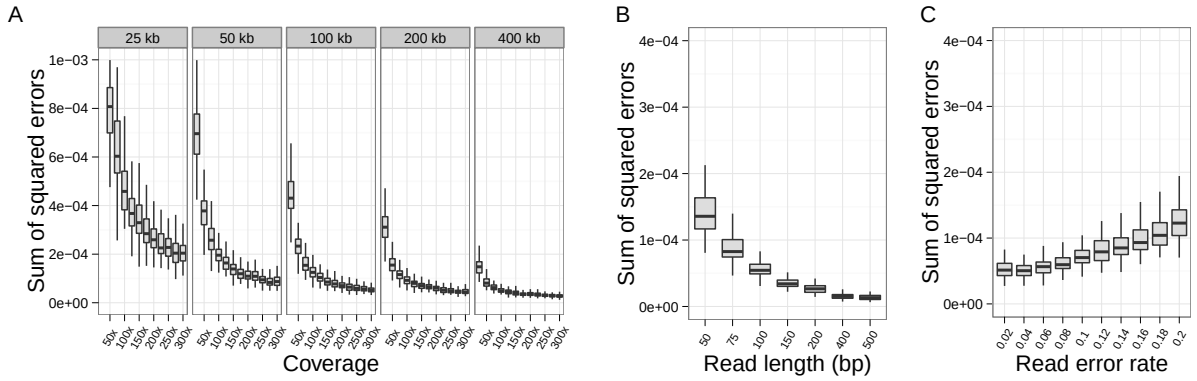

Figure 2.3: **Performance of the EM algorithm increases with coverage, region width, and read length and is robust to sequencing errors.** A) Performance of the EM algorithm increases with both coverage and width of the region used for the estimation. B) The EM algorithm performs better with longer reads, which provide more haplotype information. C) The EM algorithm maintains good performance with increasing sequence read error rate. Empirical error rates were found to be in the range of .05 - .07 errors per base call. In all simulations, we simulated paired-end pooled sequence data from 162 haplotypes at randomly drawn frequencies, with 100 replicates per parameter value level. Non-varying parameters were held at fixed values representative of our experimental data (read length 100bp, read error rate .06, coverage 200x, region width 200kb).

2.4.2 Effects of Region Width, Coverage, Read Length, and Sequencing Error

We next evaluated the performance of the EM algorithm with respect to increasing region width and coverage. In this evaluation, we simulated pooled data (100bp paired-end) from all 162 haplotypes, 100 replicates each in genomic regions ranging in size from 25kb to 400kb, at coverages ranging from 25x to 300x. We found that performance increases substantially as coverage increases, especially at the lower coverage levels (25x - 100x), and also as the region width increases (Figure 2.3A). In particular, for larger regions ($\geq$ 100kb) at moderate pooled coverage

(200x), the sum of squared errors is less than 10^{-4}, which corresponds to a root mean squared error of less than $.1\%$ per haplotype.

We also evaluated the effect of increasing read lengths on the performance of the EM algorithm. We simulated paired-end sequence data in a 200kb region with sequence read lengths ranging from 50bp to 500bp (100 replicates each). In each case, we generated 200,000 read pairs (200x coverage for 100bp reads). As expected, longer read length also increases performance, due to the additional haplotype information contained in individual reads (Figure 2.3B). Note that the effect of increased read length is equivalent to the effect of increasing SNP density, due to the fact that differences in haplotype likelihoods between strains/species are determined by the variant sites covered by the sequence reads.

Finally, we studied the effects of sequence read errors on the haplotype frequency estimation. We calculated an empirical base quality score distribution, which we shifted to obtain simulated data sets with specified error rates. In our experimental data sets, the sequence error rate calculated from the base quality scores was generally in the range of $.05 - .07$ (errors per base call), depending on the region. On simulated data sets (162 haplotypes, 200kb region, 100bp paired-end reads, 200x coverage), we found that the EM algorithm maintains good performance (sum of squared errors $\approx 10^{-4}$, average error $< .1\%$), even with error rates of 2-3x empirical error rates (Figure 2.3C).

2.4.3 Effects of Haplotype Diversity

We investigated the effects of haplotype diversity, quantified by the Shannon entropy (in natural log units) of the true haplotype frequency distribution, on the performance of the EM algorithm. We simulated pooled 100bp paired-end sequence data from 162 haplotypes at 200x coverage in a 200kb region. We generated the haplotype frequencies using symmetric Dirichlet distributions with parameter values ranging from .005 to 10, for a total of 550 replicates, which were binned by Shannon entropy (Figure 2.4). We found that the EM algorithm performs best for low entropy

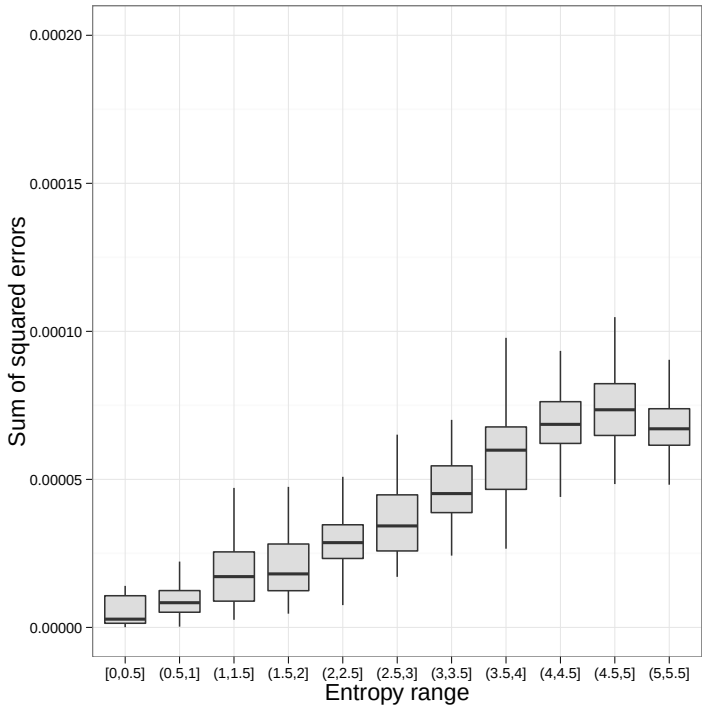

Figure 2.4: **The EM algorithm performs best when the true frequency distribution has low entropy (non-uniform, with a few haplotypes at high frequencies, with the rest at low frequencies).** The algorithm was run on simulated pooled 100bp paired-end sequence data from 162 haplotypes at 200x coverage in a 200kb region (550 replicates binned by Shannon entropy in natural log units).

frequency distributions, where there are a few haplotypes at high frequencies, with the rest at low frequencies. Performance degrades as the entropy increases, with a slight improvement for high-entropy (nearly uniform) distributions. This behavior can be explained by the fact that missing information leads to uniform estimates, which will give better results for near-uniform distributions.

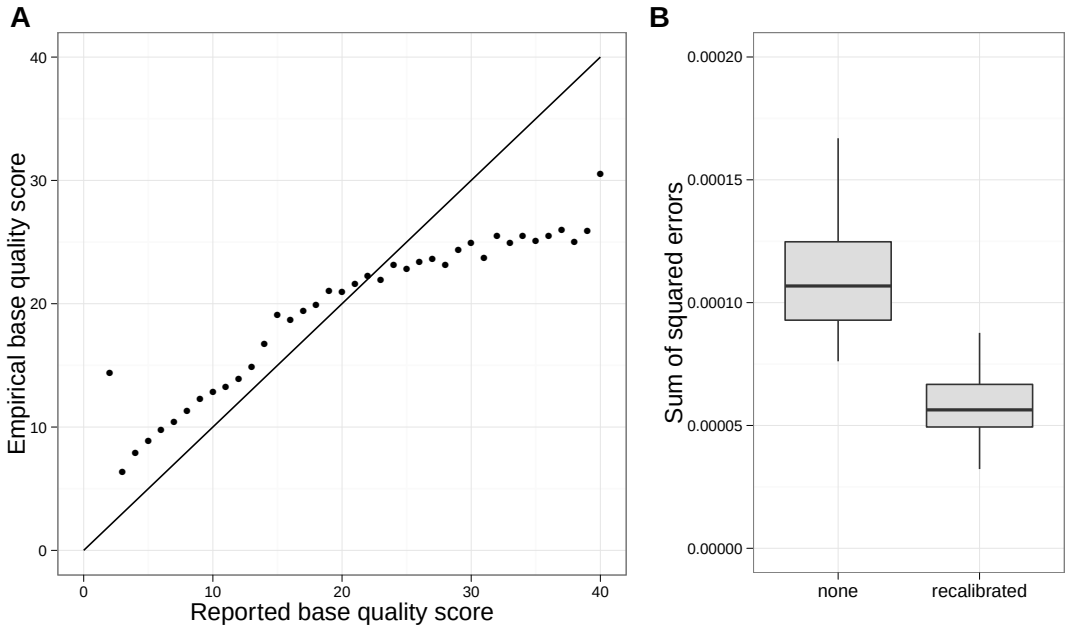

Figure 2.5: **Recalibration of base quality scores using monomorphic sites improves performance.** A) Reported base quality scores do not match empirical scores calculated from real data using monomorphic sites. B) The EM algorithm was run with and without base quality score recalibration on simulated pooled 100bp paired-end sequence data from 162 haplotypes at 200x coverage in a 200kb region (100 replicates each). Sequence errors in the simulated data were introduced with probabilities given by the empirical error rates.

2.4.4 Effects of Inaccurate Base Quality Score Reporting

The computation of haplotype likelihoods is dependent on the correct reporting and interpretation of base quality scores. By looking at monomorphic sites in our experimental data sets, we calculated an observed error rate $P_{obs}(\text{error})$ for each possible base quality score, which maps to an empirical base quality score q_{obs} according to:

$$q_{obs} = -10 \log_{10} P_{obs}(\text{error})$$

We observed that the reported base quality scores in our experimental data sets were consistently inaccurate (Figure 2.5A). This motivated the development of a recalibration method to correct for inaccurate reporting of base quality scores (see New Approaches).

In order to test our recalibration method, we simulated data sets (162 haplotypes, 100bp paired-end reads, 200kb region, 200x coverage) using the empirical error rate for each base quality score to generate sequence read errors. For each of 100 replicates, we ran the EM algorithm with and without recalibration of the base quality scores. We found the algorithm has higher accuracy with the base quality score recalibration (Figure 2.5B).

2.4.5 Random Initial Estimates To Avoid Local Maxima

We investigated the possibility that the EM algorithm could converge to non-global local maxima on the likelihood surface. We simulated data sets (162 haplotypes, 100bp paired-end reads, 200x coverage, empirical error rates) over a range of region sizes from 10kb to 200kb, starting from a uniform initial estimate in addition to a varying number of random initial estimates (0, 25, 50, 100), with 100 replicates for each combination. We found that running the algorithm multiple times with random initial estimates did not improve performance (data not shown), indicating that the EM algorithm finds the global maximum reliably starting from a uniform initial estimate.

2.4.6 Estimation of Relative Abundances of Species Based on 16S rRNA Sequence Reads

While our primary motivation for developing the haplotype inference algorithm arises in the context of *Drosophila* evolution experiments, the structure of our algorithm suggests it may be extended to the problem of inferring bacterial community composition. For instance, sequence reads derived from 16S rRNA are often used to identify the species contained within a naturally pooled sample. In the context of our method for haplotype frequency estimation, each species with a canonical 16S sequence is a known haplotype, and the challenge is to infer the haplotype frequencies based on reads copied with error from the canonical sequences. As such, our method

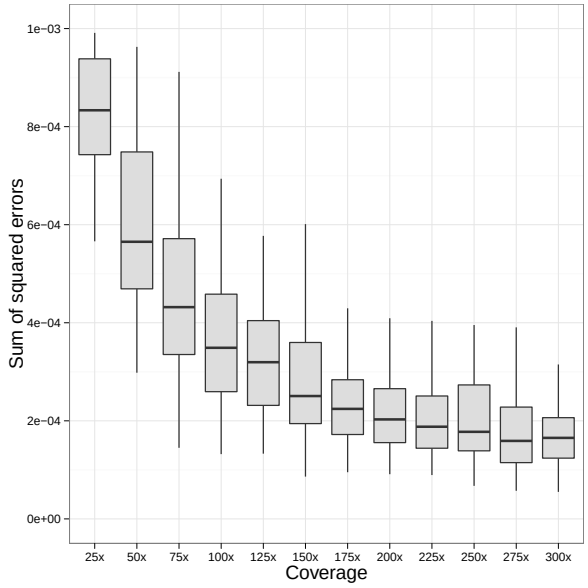

Figure 2.6: **Performance of the EM algorithm on the calculation of species-level abundances from 16S rRNA sequence data.** The algorithm was run on simulated 75bp single-end 16S sequence data from pools of 200 randomly chosen microbial species, with 100 replicates for each coverage level.

differs from most existing 16S analysis pipelines in that we do not first classify reads to species and then infer frequencies. As in the haplotype frequency inference problem, we consider the probability of each read coming from each potential source species and then iteratively converge on a set of frequency estimates using the EM algorithm.

As a preliminary test of this approach, we simulated simple communities composed of pools of 200 randomly chosen species (average 16S sequence divergence 19% ($\pm$ 2 SD of 12%)), with 75bp single-end sequence reads derived from their 16S sequences, at varying coverage levels. The simulated data reflect what one would expect from a shotgun sequencing metagenomics experiment, with sequence reads coming from random locations in the 16S sequence, from the different species according to their relative abundance within the pool. (We do not consider

the accuracy for approaches that target a specific hypervariable region, though the algorithm can be applied to such designs.) Figure 2.6 shows the performance as a function of coverage. As one example, with 150x pooled coverage of the 16S sequence, the average sum of squared errors was $\approx 2 \times 10^{-4}$, which corresponds to an average error of $.1\%$. This error is of the same order of magnitude as the error in estimation of Drosophila strain-level frequencies, which used much larger regions, but typically have lower levels of between-haplotype divergence. In these examples, we assumed the species in the pool are known. We next considered inference settings with unknown species in the pool.

2.4.7 Frequency Estimation for a Specified Set of Species Within a Larger Mixture of Unknown Species

First, because the species of some genera are better characterized than others, we investigated the utility of this method in estimating the frequencies of known species within a single genus, in a pool containing a large number of unknown species from different genera. To this end, we simulated pools containing 500 randomly chosen species: 20 from genus *Clostridium*, together with 480 species from other genera. In these simulations, the pool of known sequence reads was spiked with unknown sequence reads, with the total unknown proportion ranging from 0 to 50%, and where the unknown sequence reads were drawn uniformly at random from the 480 unknown species. The 16S average sequence divergence between pairs of known species was 12% ($\pm 10\%$) while the average divergence between known and unknown species was 20% ($\pm 6\%$).

We then estimated the frequencies of the *Clostridium* species using only the known reference sequences. We found that the EM algorithm, in combination with the haplotype likelihood filter (see New Approaches), performed well (sum of squared errors $\approx 2 \times 10^{-4}$, average error $\approx .3\%$), in spite of the large number of unknown species in the pool. (Figure 2.7).

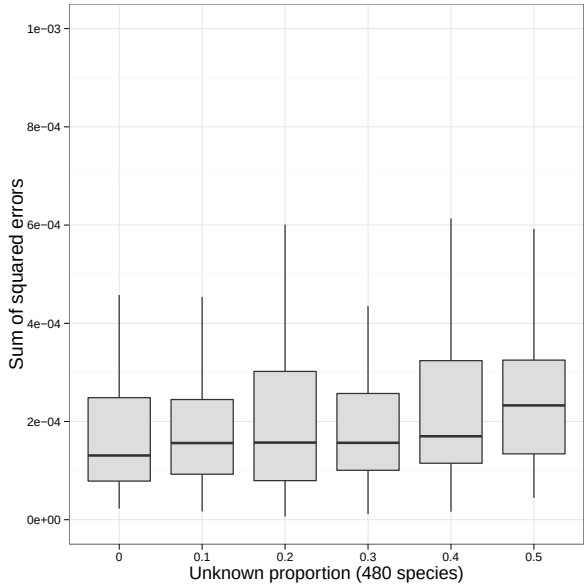

Figure 2.7: **Large numbers of unknown unrelated species do not significantly affect the estimate of within-genus species frequencies.** The EM algorithm with haplotype likelihood filter was run on simulated 75bp single-end 16S sequence reads from 500 species [20 *Clostridium* species (known), 480 non-*Clostridium* species (unknown)]. "Unknown proportion" is the total proportion of reads coming uniformly at random from the 480 unknown species, with the remainder of the reads coming from the 20 known species (100x pooled coverage, 100 replicates for each unknown proportion level).

2.4.8 Effects of Unrelated Unknown Species on Frequency Estimation

To more fully characterize the effect of unknown species on the estimation of frequencies of known species, we further investigated the scenario where there are unknown species present within the pool. For simplicity, we simulated pools with 20 known species, spiked with a single unknown species that does not belong to the same genus as any of the known species.

In this situation, reads coming from the unrelated unknown species do not map well to the reference sequences of the known species. Since the unknown is unrelated to the known species,

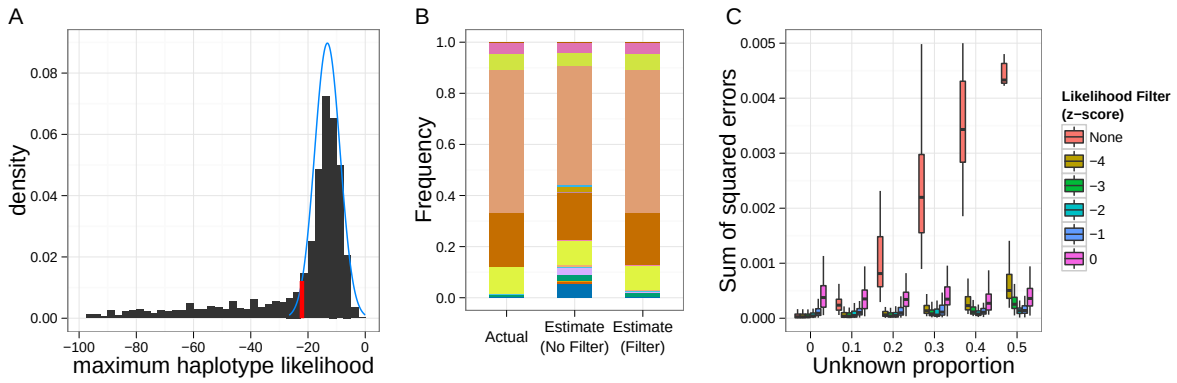

Figure 2.8: **Sequence reads from unknown unrelated species push frequency estimates toward the uniform distribution; filtering the reads based on haplotype likelihood minimizes this effect.** A) Sequence reads from unknown unrelated species have low maximum haplotype likelihoods, giving rise to the long left tail of the distribution. By calculating the theoretical distribution (blue) of maximum haplotype likelihoods based on the base quality scores from the sequence data, reads whose maximum haplotype likelihood falls below a specified threshold (red, z-score threshold = -2 in this example) can be filtered out. B) A typical example of this effect, with 50% of the reads coming from the unknown species. C) Without filtering on the haplotype likelihoods, error in frequency estimates increases with higher proportion of unknown sequence reads. 75bp single-end 16S sequence reads were simulated from 20 species, with varying proportions of reads from an unknown unrelated species (100x pooled coverage, 100 replicates per unknown proportion level).

these reads have low haplotype likelihoods across all the known species (Figure 2.8A). The end result is that the frequency estimates are pushed toward a more uniform distribution (Figure 2.8B), as the reads of the unknown species give weight to all the known species in roughly equal portions. We found that this effect can be minimized by implementing a haplotype likelihood filter (see New Approaches), which effectively keeps only those reads that come from the known species. Our simulations show that a haplotype filter z-score threshold of -2 works well to

maintain good performance in the presence of unrelated unknowns (Figure 2.8C). Without the haplotype filter, performance of the EM algorithm degrades as the proportion of the unknown species in the pool increases.

2.4.9 Effects of Related Unknown Species on Frequency Estimation

We also investigated the scenario where there is a single unknown species that is related to one of the known species in the pool. Again for simplicity, we simulated pools with 20 known species, spiked with an unknown species from the same genus as one of the known species.

We found that, in general, the sequence reads coming from the unknown species increase the estimated frequency of the known species that it is related to. However, this effect depends on the sequence similarity between the unknown and the related known species. Reads that come from a region where the two species differ significantly are filtered out and do not contribute to the estimation.

This effect is limited to the estimate of the frequency of the known species that is related to the unknown, and has little effect on the relative frequency estimates of the other species (Figure 2.9AB).

2.5 Discussion

We have presented a new method for estimating the frequencies of known haplotypes from pooled sequence data, using the haplotype information contained in individual sequence reads.

We showed that the method outperforms methods based on allele frequencies, as well as simple methods using sequence data. Using data from larger genomic regions improves the accuracy of the estimate. Increased coverage and longer read lengths also improve the performance of the algorithm. The method generally performs better for haplotype frequency distributions with lower entropy (non-uniform) than those with higher entropy (uniform), which is particu-

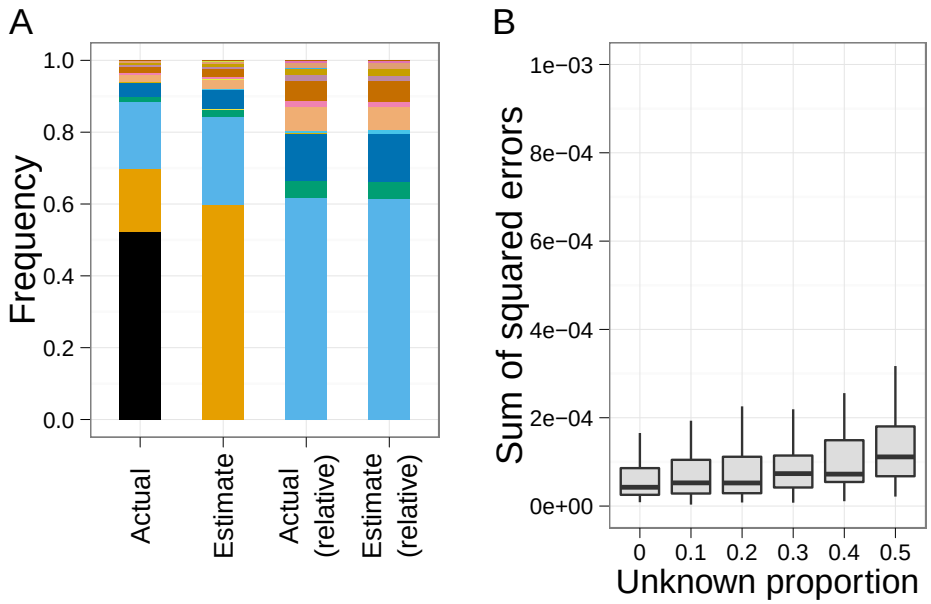

Figure 2.9: **When the pool contains an unknown species that is related to one of the known species, sequence reads from the unknown increase the frequency estimate of the most closely related known species, but have little effect on the estimation of the relative frequencies of the other known species.** A) Sequence reads from an unknown species (black) related to one of the known species (orange) contribute to the frequency estimate of that species. Shown is a typical example of this effect, with 50% of the reads coming from the unknown species. The relative abundances of the other known species are estimated accurately. B) Presence of the unknown species has little effect on the estimation of the relative frequencies of the other (unrelated) known species. 75bp single-end 16S sequence reads were simulated from 20 known species and 1 unknown related species (100x pooled coverage, 100 replicates for each unknown proportion level).

larly notable in metagenomics contexts, where species/strain abundances are far from uniform. The method incorporates uncertainty in the sequence reads by using the reported base quality scores. Recalibration of base quality scores using monomorphic sites in the pooled data leads to

better performance.

We note that in the EM algorithm, as well as the two simple approaches, haplotype information from individual sequence reads is combined across a genomic region, which minimizes the effects of local areas of low coverage. By contrast, single-site allele frequency estimates have larger variance at sites with lower coverage, which decreases the accuracy of methods based on these estimates.

The method relies on the probabilistic assignment of sequence reads to the known haplotypes. The method works best when the SNP density (the number of SNPs per base pair) is high; ideally individual read pairs will cover multiple SNPs. In the DGRP Drosophila strains, the SNP density is $\sim 1/20$ SNP/bp so that 100bp paired-end reads contain on average 10 SNPs per pair, which is sufficient for this probabilistic assignment. As sequence read lengths increase with advances in sequencing technology, we anticipate that this method will be useful for a wide variety of organisms.

This method can be improved to handle more complicated genomic situations. For example, the probability model described could be enhanced with known information about between-species or between-strain copy number variation. In addition, because estimation at the species level (or above) requires mapping the reads to multiple references, there is room for improvement in the efficiency of this step, including methods for mapping reads to multiple references simultaneously with automatic calculation of haplotype likelihoods, which can then be input directly into the EM iteration.

This method has immediate application in the analysis of pooled data from artificial selection experiments where the founding haplotypes are known. In essence, by using information from the founding population, we are able to infer haplotype information about the final pooled population, which has previously only been available when individuals have been sequenced separately. This haplotype information can then be used for various purposes (e.g. to look for signatures of selection).

It should be noted that in the experimental evolution setting, the haplotype frequency estimates obtained are local, and will vary across the genome due to recombination over the course of the experiment. In the case where a recombination has occurred within the region under consideration, nearly all sequence reads coming from the recombined haplotype will come from one or the other of the two original haplotypes. Because of this, reads from the recombined haplotype will contribute to the frequency estimates of both of the original haplotypes, with the exact proportion determined by the location of the recombination point within the region. The few reads that span a recombination point will have very low haplotype likelihoods, and will contribute neglibly to the final estimate. In practice, one can choose the width of the genomic region used for the estimation to be smaller than the expected length scale of recombination. For example, in Drosophila selection experiments lasting 25 generations, we expect to see recombination breakpoints at a scale of $\approx$ 1mb, whereas our method obtains very accurate results with much smaller regions of ≈ 100 kb.

Haplotype frequency estimation from pools may also be useful in QTL mapping studies. In studies with recombinant inbred lines, several generations of inbreeding are carried out with lines that are derived from mixed populations founded by multiple strains, and the parent of origin for each segment in each inbred line is inferred and correlated with phenotypic trait values. As an alternative, it may be possible to perform haplotype frequency estimation from pooled sequencing of the mixed populations directly and map traits by correlating haplotype frequencies with trait values.

In addition to applications in experimental evolution settings with known founder haplotypes, we also explored whether the method may have utility for estimating the relative abundances of known species from metagenomic data. We specifically tested the setting in which one has short-read shotgun sequencing data from 16S rRNA, rather than amplicon-based 16S rRNA sequences. The setting we have tested is relevant to the practical setting in which one generates whole-genome metagenomic data and then focuses on reads from 16S rRNA for quantification

of known species. In our simulation experiments, we have explored various factors affecting performance. We note how having closely related species in the sample that are not in the reference set will elicit an implicit "clustering" behavior, in which the most closely related known species in the reference set will have its frequency inflated (Figure 2.9A). The relative frequencies of unrelated species are not affected however (Figure 2.9B). This behavior extends nicely to the case where one is interested in the relative frequencies of species from a well characterized genus (e.g. one in which most member species have known canonical 16S rRNA sequences). In this case, we show the relative frequencies can be estimated well, even in the presence of hundreds of background species (Figure 2.7). In these cases, our haplotype likelihood filter acts as a form of clustering tool in that we only base the frequency estimates on reads that are close to the taxa of interest.

Just as SNP density is highly relevant in the haplotype inference problem, a key factor in the performance for metagenomic applications is the amount of 16S sequence divergence between the taxa of interest. Our examples were based on species sets whose divergences were generally between 5% and 33% and the method performed well in these settings. More closely related species will be more difficult to distinguish, and at an extreme, strains within species will require more than the 16S sequence to distinguish. One limitation is that our method infers frequencies of taxa represented by specific canonical rRNA sequences. How our method would perform with a single canonical sequence per genus, class, or phylum is not clear, and future extensions of this work may seek to specifically alter the underlying probability model to include diversity within taxa to specifically address this problem.

The potential advantage over existing methods for handling 16S data lie in the method's ability to handle base quality scores explicitly and give probabilistic weights to the source of reads, rather than making hard classifications and estimating frequencies based on those classifications. As we have shown in the context of haplotype frequency estimation (Figure 2), such estimation performs much worse than the probabilistic approach we take here.

The software implementing the method, `harp`, is open source and available for download, and can be easily integrated into existing analysis pipelines.

2.6 Material and Methods

2.6.1 Implementation

We implemented both simple approaches and the EM algorithm described above in a C++ program called `harp` (*Haplotype Analysis of Reads in Pools*). The program takes as input a standard BAM file with mapped sequence reads, a reference sequence in FASTA format, and known haplotypes in the SNP data format used by the DGRP project (Mackay *et al.*, 2012). Alternatively, to estimate abundances of distinct species within a pool, the program will accept multiple BAM files, each paired with the reference sequence used for the read mapping. The software uses the `samtools` API for random access to BAM files (Li *et al.*, 2009). The program includes many options for the user to customize the analysis, including choice of algorithm, the genomic region to analyze, parameters for sliding windows within the region, convergence threshold for the EM algorithm, parameters used to generate multiple random initial estimates to avoid local maxima, base quality score recalibration, and haplotype likelihood filtering threshold. The program also calculates standard errors for the haplotype frequency estimates, using general properties of the EM algorithm and maximum likelihood estimators (details on this calculation below).

2.6.2 Performance Evaluation

We used the following procedure to simulate pooled sequence data:

- Draw a random haplotype frequency distribution (the *true* distribution) from a symmetric Dirichlet distribution. The symmetric Dirichlet distribution is parameterized with a single parameter α that governs the uniformity of the randomly drawn frequency distribu-

tions: $\alpha = 1$ gives an identical probability to each possible frequency distribution, $\alpha > 1$ generates frequency distributions that are close to uniform (i.e. all haplotypes at similar frequencies), and $\alpha < 1$ generates frequency distributions that are more non-uniform (i.e. a few common haplotypes, and many rare ones). The parameter value $\alpha = .2$ was chosen to produce frequency distributions with non-uniformity similar to that observed in our experimental *Drosophila* data.

- Draw random sequence reads by choosing the haplotype according to the true distribution and the starting position uniformly at random over the given genomic region. For paired-end reads, the paired end distance was chosen according to a Poisson distribution fitting the experimental *Drosophila* data.

- For segregating sites denoted by ambiguous base codes, draw allele frequencies according to a symmetric Dirichlet distribution. Choose the true base at a segregating site according to the allele frequencies. (For biallelic sites denoted by a 2-base ambiguous code, e.g. R for A or G, we set the Dirichlet parameter $\alpha = 1$, i.e. the allele frequency was chosen uniformly at random. For sites denoted by N (any) in the haplotype, we set $\alpha = .1$, as we expect that most of these sites have an allele that is at or near fixation).

- Generate base quality scores according to the empirical distribution obtained from the real data (either *Drosophila* or 16S), and introduce sequencing errors with error rates determined by the base quality scores.

For the simulated *Drosophila* pooled sequence data, we generated mapped read files (SAM format), which were subsequently converted to binary format (BAM) using `samtools` (Li *et al.*, 2009). For the simulated 16S rRNA sequence data, we generated raw read files (fastq format), which were then mapped to reference sequences using `bwa` (Li and Durbin, 2010).

For our performance metric, we used the sum of squared errors between the true haplotype frequencies of the simulated data and the frequencies estimated by the EM algorithm:

$\sum_{h=1}^{H} \left(f_h^{true} - f_h^{estimated}\right)^2$, in addition to the root mean squared error. We define f_{true} to be the realized frequencies of the haplotypes in the sequenced reads from the DNA pool. We do this to quantify the error of estimation, rather than the stochasticity of the pooled sequencing. For the purposes of comparison across simulations, we report the sum of squared errors in the figures. However, we also give the equivalent root mean squared errors in the main text for interpretation.

2.6.3 Simulation of *Drosophila* Pooled Sequence Data

To evaluate the performance of the algorithms, we used simulated pooled sequence data based on experimental data from selection experiments in *Drosophila melanogaster* (Turner and Miller, 2012). The data consisted of Illumina 85bp and 100bp paired-end sequence reads generated from 4 pools of 120 *D. melanogaster* individuals each, sequenced at 200x average coverage. For our known haplotypes, we used the publicly available SNP data from 162 Drosophila inbred lines representing Freeze 1 of the DGRP project. The published haplotypes include ambiguous base codes (e.g. R for A or G) to represent sites that have multiple alleles still segregating within the inbred line. The ambiguous base code N is used at sites where there was not enough sequence data to make a base call.

2.6.4 Simulation of 16S rRNA Pooled Sequence Data

To evaluate the ability of the EM algorithm to estimate species abundances from a pool of 16S rRNA sequences, we used sequences from the Greengenes 2011 release of its 16S rRNA sequence database (DeSantis *et al.*, 2006), which consists of approximately 800,000 sequences. We simulated 75bp single-end sequence reads with an empirical base quality score distribution based on the publicly available Illumina-sequenced "mock community" short read archive datasets from the Human Microbiome Project (NIH HMP Working Group, 2009). Specific details on number of species used in each simulated experiment are described in the results.

2.6.5 Formal EM Calculation

Expectation step: We calculate the expectation of the complete data log-likelihood, where the expectation is taken over the posterior distribution of the missing data given the observed data and the current estimate. Recall that $p_{j,h} = P(\eta_j = h|r_j, f^{(i)})$ is the probability that read r_j came from haplotype h, given our current haplotype frequency estimate $f^{(i)}$.

$$\begin{aligned}
Q(f|f^{(i)}) &= E_{\eta|r,f^{(i)}}[\log L(f|\eta, r)] \\
&= E_{\eta|r,f^{(i)}} \sum_j \log P(\eta_j, r_j|f) \\
&= \sum_j E_{\eta_j|r_j,f^{(i)}} \log P(\eta_j, r_j|f) \\
&= \sum_j \sum_h P(\eta_j = h|r_j, f^{(i)}) \log P(h, r_j|f) \\
&= \sum_j \sum_h p_{j,h}[\log P(r_j|h) + \log P(h|f)] \\
&= \sum_j \sum_h p_{j,h} \log f_h + C \\
&= \log \prod_h f_h^{\sum_j p_{j,h}} + C
\end{aligned}$$

where C is a constant independent of f.

Maximization step: Our next estimate $f^{(i+1)}$ is given by the f that maximizes the expected log-likelihood:

$$f^{(i+1)} = \underset{f}{\mathrm{a}rgmax}\, Q(f \mid f^{(i)})$$

First note that the function $R(f) = \prod_h f_h^{\alpha_h}$ is maximized by $f_h = \alpha_h / \sum_i \alpha_i$. (For example, the maximum likelihood estimator for the parameters of a multinomial distribution is given by

the vector of count proportions.)

Since $\log$ is monotonic and

$$\sum_h \sum_j p_{j,h} \;=\; \sum_j 1 \;=\; N \text{ (the number of reads)},$$

$Q(f|f^{(i)})$ is maximized when, for all h:

$$f_h = \frac{\sum_j p_{j,h}}{N}.$$

In other words, our next estimate $f^{(i+1)}$ is given by the average of the posterior vectors:

$$f^{(i+1)} = \frac{\sum_j p_j}{N}$$

2.6.6 Calculation of Standard Errors

We use general properties of the EM algorithm and maximum likelihood estimators to calculate standard errors for our haplotype frequency estimates, following Lange (2010). For brevity, we let $L(f)$ be the log-likelihood of f. Our strategy to calculate standard errors of our maximum likelihood estimate $\hat{f}$ is as follows:

1. estimate the observed information $I = -d^2 L(\hat{f})$
2. $\hat{f}$ is asymptotically normal with covariance matrix I^{-1}, so the standard error estimates are the square roots of the diagonals of I^{-1}

One slight complication is that our estimate $\hat{f}$ is subject to a linear constraint (the frequencies must sum to 1). Below, we also show how to adjust this calculation to handle this constraint.

2.6.7 Calculating d^2L

Let g be the minorizing function for the EM algorithm:

$$g(f|f_0) = Q(f|f_0) + L(f_0) - Q(f_0|f_0)$$

Then g satisfies the relations for all f, f_0:

$$g(f|f_0) \leq L(f)$$
$$g(f_0|f_0) = L(f_0)$$

Note that $\nabla g(f|f_0) = \nabla Q(f|f_0)$. Also, $L(f) - g(f|f_0)$ is minimized $f = f_0$, so $\nabla L(f) - \nabla g(f|f_0) = 0$ at $f = f_0$. This means that $\nabla L(f_0) = \nabla Q(f_0|f_0)$. Note that $\nabla Q(f_0|f_0)$ is the gradient $\nabla Q(f|f_0)$ computed as a function of f, then evaluated at $f = f_0$.

In summary, we can write the score function $S = (S_1, \ldots, S_H)'$, defined to be the gradient of the log-likelihood, as:

$$S(f) = \nabla L(f) = \nabla Q(f|f)$$

Since $dS = d^2L$, we need to find the partial deriviatives of S.

Recall that we have:

$$Q(f|f_0) = \sum_h (\sum_j p_{j,h}) \log f_h$$

where $p_{j,h}$ is the haplotype posterior vector, representing the probability that read r_j came from haplotype h.

Note that the $p_{j,h}$ depend on f_0, but not f, so the partial derivatives of $Q(f|f_0)$ have a simple

form:

$$\frac{\partial Q(f|f_0)}{\partial f_h} = \frac{\sum_j p_{j,h}(f_0)}{f_h}$$

Now we evaluate at $f = f_0$, and drop the subscript:

$$S_h(f) = \frac{\sum_j p_{j,h}(f)}{f_h}$$

We can now compute partial derivatives of the score function:

$$\frac{\partial S_h}{\partial f_k} = \frac{1}{f_h} \sum_j \frac{\partial p_{j,h}}{\partial f_k} - \frac{\mathbf{1}_{k=h}}{f_h^2} \sum_j p_{j,h}$$

To compute the partial derivatives of $p_{j,h}$, first write $p_{j,h}$ as:

$$p_{j,h} = \frac{l_{j,h} f_h}{P_j}$$

where $P_j = \sum_h l_{j,h} f_h$ is the total probability of read r_j. Since $\frac{\partial P_j}{\partial f_k} = l_{j,k}$, we have:

$$\begin{aligned} \frac{\partial p_{j,h}}{\partial f_k} &= \frac{l_{j,h}}{P_j} \mathbf{1}_{k=h} - l_{j,h} f_h \frac{1}{P_j^2} \frac{\partial P_j}{\partial f_k} \\ &= \frac{l_{j,h}}{P_j} \mathbf{1}_{k=h} - \frac{l_{j,h} l_{j,k} f_h}{P_j^2} \end{aligned}$$

2.6.8 Adjusting for the linear constraint

We continue to follow Lange (2010) to handle the linear constraint $\sum_h f_h = 1$. Let $V = \mathbf{1}_H^t = (11 \cdots 1)$ be the row vector with H 1's, so we can write our constraint as $Vf = 1$.

We let W be a matrix with $H-1$ column vectors orthogonal to V:

$$W = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 \end{pmatrix}$$

We reparametrize by β, using the relation $f = \alpha + W\beta$, where α satisfies the constraint $V\alpha = 1$. As a function of β, the log-likelihood $L(\alpha + W\beta)$ has observed information $-W^t d^2 L(\alpha + W\beta)W$, which gives an estimate $\text{Var}(\hat{\beta}) = -[W^t d^2 L(\hat{f})W]^{-1}$. This gives the estimate $\text{Var}(\hat{f}) = \text{Var}(W\hat{\beta}) = -W\,[W^t\, d^2 L(\hat{f})\, W]^{-1}\, W^t$, where $d^2 L(\hat{f})$ was estimated above.

2.6.9 Haplotype Likelihood Filter Threshold

In this section we show how we approximate the mean and variance of the maximum haplotype likelihood distribution, which are used to calculate the threshold for the haplotype likelihood filter.

The main idea is that the maximum haplotype likelihood $P(r|\eta, q)$ for a read is usually attained by the actual haplotype from which the read was copied. Because of this, we can approximate the mean and variance of the maximum haplotype likelihood distribution by the mean and variance of the distribution of $P(r|\eta = h, q)$ for reads r copied from a fixed haplotype h, under the empirical distribution of the base quality scores q. For ease of calculation and implementation, we actually calculate the statistics for the log-likelihood distribution $\log P(r|\eta = h, q)$. We also suppress the dependence on h in the following for ease of notation.

We assume we have an empirical per-position base quality score distribution obtained from the data, where for simplicity we assume that the base quality scores at different positions are independent of each other. So for reads of length L, we have discrete random variables

$(Q_1, \ldots, Q_L)$. For example, for Illumina reads, Q_i will take values in $\{0, \ldots, 40\}$. For a base quality score q, we let $\epsilon(q)$ denote the probability of a read error for that base. Typically we will have the Phred-encoded error probabilities, so $\epsilon(q) = 10^{-q/10}$.

Let $(h_1, \ldots, h_L)$ denote the true haplotype sequence from which the read is copied, $(q_1, \ldots, q_L)$ denote the base quality scores, and $(r_1, \ldots, r_L)$ denote the read bases. Note the slight change in notation from the description of the probability model (New Approaches): here the subscript denotes position in the sequence read.

By our assumption of position independence, we have $\log P(r|q) = \sum_{i=1}^{L} \log P(r_i|q_i)$. Note that each of these terms is random:

$$\log P(r_i|q_i) = \begin{cases} \log(1 - \epsilon(q_i)) & \text{w/probability} \quad 1 - \epsilon(q_i) \\ \log(\epsilon(q_i)/3) & \text{w/probability} \quad \epsilon(q_i) \end{cases}$$

where the first event corresponds to $r_i = h_i$ (no error), and the second event represents the 3 possible error bases when $r_i \neq h_i$, each one occuring with probability $\epsilon(q_i)/3$.

Next we calculate the conditional expectation of $\log P(r_i|q_i)$ given q_i:

$$E[\log P(r_i|q_i) \mid q_i] = [1 - \epsilon(q_i)] \log(1 - \epsilon(q_i)) + \epsilon(q_i) \log(\epsilon(q_i)/3)$$

Note that this expression does not depend on r_i. Now we can take the expectation over Q_i:

$$\begin{aligned} E[\log P(r_i|q_i)] &= EE[\log P(r_i|q_i) \mid q_i] \\ &= \sum_{q_i} E[\log P(r_i|q_i) \mid q_i] \;\; P(Q_i = q_i) \end{aligned}$$

Note that this expression doesn't depend on either r_i or q_i, and is a function solely of the empirical distribution Q_i. Finally, we can sum over positions to obtain:

$$E[\log P(r|q)] = \sum_{i=1}^{L} E[\log P(r_i|q_i)]$$

which represents the expected log haplotype likelihood for a read r corresponding to the true haplotype h, under the empirical base quality distribution.

Similarly, we calculate $E[(\log P(r_i|q_i))^2]$ for each position, from which we can obtain the per-position variance Var$[\log P(r_i|q_i)]$. Again by our position-independence assumption, we have:

$$\text{Var}[\log P(r|q)] = \sum_{i=1}^{L} \text{Var}[\log P(r_i|q_i)]$$

We use the calculated mean and variance of $\log P(r|q)$ to translate a user-defined z-score threshold into a haplotype log-likelihood threshold for filtering out low-likelihood reads.

CHAPTER 3

forqs: Forward-in-time Simulation of Recombination, Quantitative Traits, and Selection

3.1 Abstract

`forqs` is a forward-in-time simulation of recombination, quantitative traits, and selection. The `forqs` simulator was designed to investigate haplotype patterns resulting from scenarios where substantial evolutionary change has taken place in a small number of generations due to recombination and/or selection on polygenic quantitative traits. To do this, `forqs` tracks individual haplotype chunks during the course of the simulation; each chunk carries an identifier specifying the ancestral individual from which the block is derived. `forqs` is implemented as a command-line C++ program, using a modular design that gives the user great flexibility in creating custom simulations.

3.2 Introduction

Simulations have a long history in population genetics, both for verifying analytical results and for exploring population models that are mathematically intractable. Population genetics simulations can be broadly classified as forward-in-time (e.g. Wright-Fisher) or backward-in-time (e.g. coalescent). Coalescent simulations (e.g. `ms` (Hudson, 2002), `MaCS` (Chen *et al.*, 2009), `fastsimcoal` (Excoffier and Foll, 2011)) are very efficient for simulating neutral sequence

data because they only need to track lineages that are ancestral to the sample. While it is possible to simulate certain selection scenarios within the coalescent framework (Hudson and Kaplan, 1988; Ewing and Hermisson, 2010), one must turn to forward-in-time simulations to model selection in a flexible way.

Many forward-in-time simulators are currently available. Most of these simulators use a mutation-centric approach, implemented by storing the mutations carried by individuals in an array. To handle selection, the majority of these simulators assign selection coefficients to individual mutations (e.g. `ForwSim` (Padhukasahasram *et al.*, 2008) `Fregene` (Chadeau-Hyam *et al.*, 2008) `GENOMEPOP` (Carvajal-Rodriguez, 2008) `SFS_CODE` (Hernandez, 2008) `TreesimJ` (O'Fallon, 2010), `SLiM` (Messer, 2013)), while a few also include support for quantitative traits (e.g. `ForSim` (Lambert *et al.*, 2008), `quantiNemo` (Neuenschwander *et al.*, 2008), `simuPOP` (Peng and Kimmel, 2005)). Hoban *et al.* (2011) and Yuan *et al.* (2012) are recent reviews providing a comprehensive comparison of these and other simulators.

In many scenarios of biological interest, substantial evolutionary change has taken place in a small number of generations due to recombination and/or selection on standing variation, rather than mutational input. For example, one may be interested in the genome-wide haplotype patterns that emerge from admixture between historically isolated populations (Wegmann *et al.*, 2011), or from artificial selection on a quantitative trait. Studying these haplotype patterns can be difficult with existing forward-in-time simulators, because detailed information about the mosaic haplotype structure of individuals is not readily available, and must be inferred from the output sequences of the simulation and/or stored recombination event data. In addition, forward-in-time simulators that store entire sequences incur a severe trade-off between the size of the genomic regions and the size of the populations simulated.

Motivated by such examples, we have implemented a new forward-in-time simulation approach that, instead of tracking single-site variants, tracks individual haplotype chunks as they recombine over multiple generations. Further, we have designed the simulator for fast simulation

of quantitative traits under selection. We have labeled this software `forqs` (Forward-in-time simulation of Recombination, Quantitative Traits, and Selection). Similar approaches have been implemented recently by Haiminen *et al.* (2013) and by Aberer and Stamatakis (2013), for the simple selection models with per-mutation fitness effects.

The haplotype-based design allows for fast simulation of whole genomes, with very efficient memory usage. For example, `forqs` can easily simulate two populations (size 10,000 each) selected for different optimal trait values, where individuals have human-sized genomes (23 chromosome pairs, 100 mbp each), taking $\sim$2 seconds/generation. For comparison, existing forward simulators are limited by the amount of sequence that can be stored in arrays in memory: for the above 20,000 individuals, 16GB of memory would permit the storage of only 3.2 million base pairs of sequence per individual, which is an order of magnitude smaller than the smallest human chromosome. `forqs`' design also preserves information about the haplotype structure of individuals, which allows for immediate identification of genomic regions where individuals share identical-by-descent haplotype tracts.

Our simulator uses a modular architecture to allow the user to flexibly specify recombination maps, mutation rates, demographic models, quantitative traits and fitness functions. This modular approach facilitates simulation of complicated scenarios and investigation of the resulting haplotype patterns. `forqs` is currently under active development to support ongoing projects.

3.3 Design and Implementation Summary

`forqs` begins with a set of founding haplotypes carried by the individuals in the initial generation. Individuals are diploid, and carry a user-specified number of chromosome pairs. By assigning a unique identifier to each founding haplotype, individual haplotype chunks are tracked as they recombine over subsequent generations (Figure 3.1). For the purposes of simulation, any existing neutral variation on the haplotype chunks can be ignored, and only those loci with

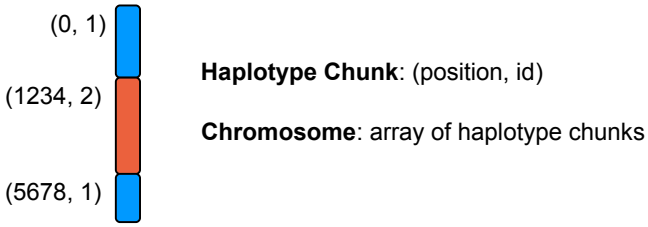

Figure 3.1: **`forqs` chromosome representation**. An individual chromosome is represented by a list of haplotype chunks. Each haplotype chunk is represented by two numbers *(position, id)*: the position where it begins, and the identifier of the founding haplotype from which it is derived. This cartoon depicts a chromosome with 3 haplotype chunks as the result of recombination (double crossover) between two founder chromosomes.

fitness effects need to be tracked.

`forqs` performs the following actions during a single cycle of the simulation: 1) generation of new populations, 2) genotyping, 3) quantitative trait evaluation, 4) fitness evaluation, and 5) reporting. `forqs` has a flexible design in which the simulator delegates specific tasks or calculations to configurable modules. The user specifies which modules to instantiate in a configuration file.

In addition to the primary modules that are used to specify demography, mutation, recombination, quantitative traits, fitness, and reported output, there are several building block modules that provide basic functionality to the primary modules. For example, `Trajectory` modules provide a unified method for specifying values that change over time, such as population sizes or migration rates. Similarly, `Distribution` modules can be used to specify how to draw particular random values (e.g. QTL positions or allele frequencies).

As an illustration of `forqs` configuration, suppose that a user wanted to simulate populations undergoing neutral admixture. The user would specify a `PopulationConfigGenerator` module representing a stepping stone or island model with the desired population size and migration rate trajectories. However, the user would not specify any quantitative

trait modules, and would use the default `FitnessFunction` module that assigns identical fitness values to all individuals. On the other hand, to simulate an artificial selection experiment with truncation selection on a single quantitative trait, the user would specify the trait with quantitative trait loci (QTLs) and effect sizes, and choose a `FitnessFunction` module that selects the desired proportion of individuals to produce the next generation. Alternatively, the user could indicate that the QTLs and effect sizes should be drawn randomly from user-specified distributions.

The representation of chromosomes as haplotype chunks in `forqs` makes very efficient use of memory, independent of the size of the chromosomes. On a typical laptop computer, for a population size of 1 million, simulations take $\sim$1.5 sec/gen for neutral simulations and $\sim$3 sec/gen with selection at a single locus. Decreasing the population size allows the simulation of a greater number of generations in a reasonable amount of time: a population size of 10,000 takes $\sim$3 sec per 100 generations (without selection, with a slight increase with selection). However, `forqs`' design comes with the tradeoff that memory usage grows linearly with the number of generations simulated, due to recombination. Thus, for investigations focusing on mutational input over a large number of generations (e.g. studies involving demographic changes taking place over thousands of generations), `forqs`' design is not as efficient as array-based implementations (e.g. `SLiM` or `SFS_CODE`) that were designed specifically for these scenarios. Similarly, we recommend that `forqs` be used in conjunction with a coalescent simulator to generate neutral variation, rather than running `forqs` for a long burn-in period to reach mutation-drift equilibrium.

`forqs` has been extensively tested for correctness, both at the level of individual code units, and in its large-scale behavior in comparison to theoretical predictions from population genetics and quantitative genetics. Validation results, tutorials and documentation can be found in the Supplementary Information. Configuration files for all simulations mentioned in this manuscript are included in the `forqs` software packages.

3.4 Getting Started

forqs binaries (Linux, OSX, and Windows) are distributed in zip file packages containing the executables, documentation, and example configuration files. The latest forqs packages are available here:

```
https://bitbucket.org/dkessner/forqs/downloads
```

forqs is a command line program that takes a single argument specifying the configuration file for the simulation:

```
forqs config_file
```

The forqs packages include an examples directory containing several example configuration files. After unzipping the forqs package, you can run forqs directly from the package directory:

```
bin/forqs examples/example_1_locus_selection.txt
```

forqs puts all output files in the output directory specified in the configuration file. After running this command, you will find a new directory output_example_1_locus_selection with the output from this simulation. For convenience, you can put the forqs executable somewhere in your PATH, e.g. ~/bin, so that you don't have to type the path when running the program.

In addition to the document you are now reading, forqs packages include the forqs Module Reference (docs/forqs_module_reference.html) which contains details about the configurable modules available to the user, including parameter names, usage, and links to examples (Figure 3.2).

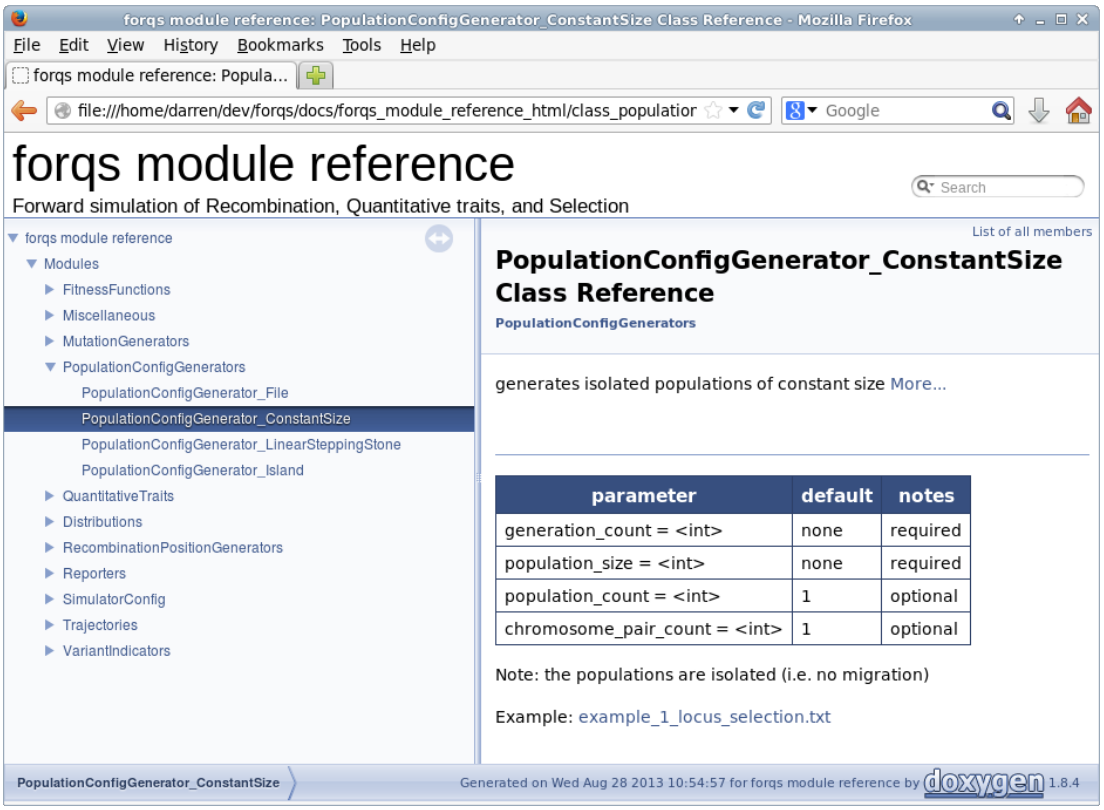

Figure 3.2: **`forqs` Module Reference screenshot**.

3.5 Modular Architecture

`forqs` begins with a set of founding haplotypes representing the individuals in the initial generation. By assigning a unique identifier to each founding haplotype, individual haplotype chunks are tracked as they recombine over subsequent generations. For the purposes of simulation, any existing neutral variation on the haplotype chunks can be ignored. When simulating selection on standing variation, only those loci with fitness effects need to be tracked.

Internally, `forqs` represents an individual chromosome as a list of haplotype chunks (Figure 3.1). Individuals are diploid, and carry a user-specified number of chromosome pairs. To represent genetic variation at particular loci, `forqs` queries functions called

`VariantIndicators` to obtain the variant values carried by the founding haplotypes at those loci.

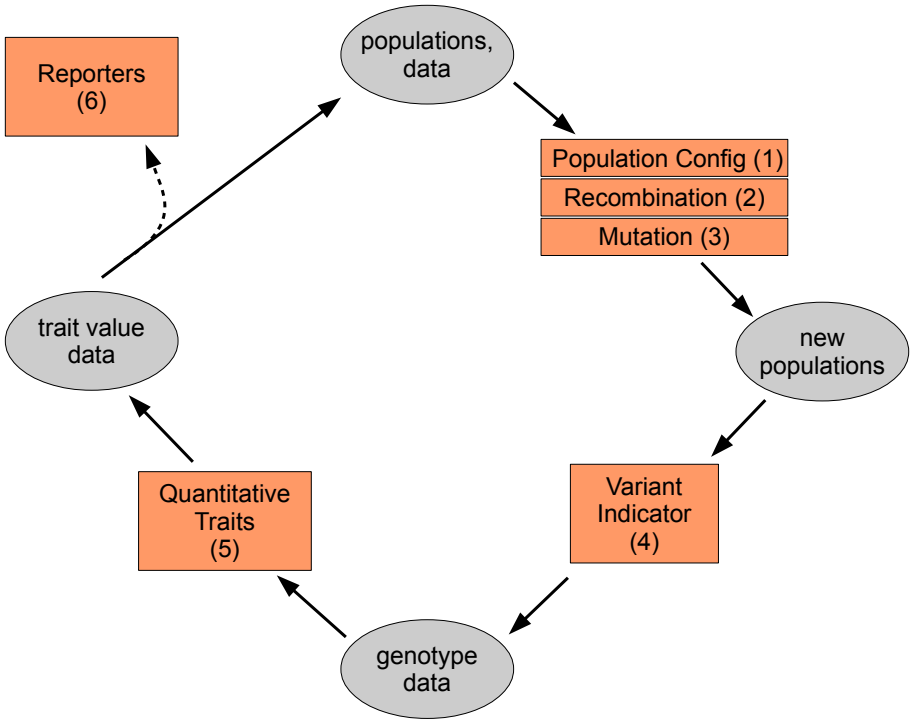

Figure 3.3: **`forqs` modular design**. The orange boxes represent places where the user can plug in configurable modules.

`forqs` uses configurable *modules* (Section 3.7) that are plugged into the main simulator to change the behavior of the simulation (Figure 3.3). `forqs` performs the following actions during a single cycle of the simulation:

(1-3) Generation of new populations: The `PopulationConfigGenerator` module (1) provides the simulator with a *population configuration* that specifies how to create individuals in the next generation from parents in the previous generation, based on user-specified fitness functions, population size and migration rate trajectories. Recombination and mutation are handled by a user-specified `RecombinationPositionGenerator`

module (2) and/or `MutationGenerator` module (3), respectively.

(4) Genotyping: individuals are "genotyped" at a set of loci, using the `Variant-Indicator` module to obtain variant (SNP) values for each individual. The list of loci to genotype is determined from the quantitative traits and reporters that are specified by the user.

(5) Quantitative trait evaluation: for each quantitative trait, a trait value is calculated for each individual, based on the individual's genotype at loci affecting the trait. Each quantitative trait is specified by a `QuantitativeTrait` module.

(6) Reporting: each `Reporter` module updates its output files using current information from the populations (which includes genotypes, trait values, and fitnesses).

Each configurable module in Figure 3.3 is actually an interface with multiple implementations. The user specifies which module implementations to instantiate in a `forqs` configuration file. More details on the primary module interfaces and their interaction with the simulator can be found in Section 3.7.4. The `forqs` Module Reference contains detailed information on the primary module implementations available to the user.

In addition to the primary modules shown in the diagram, there are several building block modules (Section 3.7.5) that provide basic functionality to the primary modules. For example, `Trajectory` modules provide a unified method for specifying values that change over time, such as population sizes or migration rates. Similarly, `Distribution` modules can be used to specify how to draw particular random values, for example, QTL positions or allele frequencies.

3.6 Tutorial Introduction

In this section we introduce `forqs` configuration files, starting with a minimal example and gradually adding modules to add complexity to the simulation. All example configuration files can be found in the `examples` directory of the `forqs` package.

3.6.0 A minimal example

We start with a minimal example configuration file:

```
#
# tutorial_0_minimal.txt
#

PopulationConfigGenerator_ConstantSize pcg
    population_size = 10
    generation_count = 3

SimulatorConfig
    output_directory = output_tutorial_0_minimal
    population_config_generator = pcg
```

There are two modules specified in this configuration file. The first module is `PopulationConfigGenerator_ConstantSize`, which we have assigned the object id `pcg`. The object id, which can be any text string, allows other objects to reference this object. We have set two parameters for this module, which specify the population size (`population_size = 10`) and the number of generations to simulate (`generation_count = 3`).

The second module is `SimulatorConfig`, the top-level module. This module must be specified last in the configuration file, because it has references to the primary modules (the general rule is: objects must be specified before they can be referenced by other objects). In this case, we are telling `SimulatorConfig` to use object `pcg` as our `PopulationConfigGenerator`, which is the only primary module that is required to

be specified. The other primary modules have trivial defaults, and we leave them unspecified. In addition to the primary module references, `SimulatorConfig` also contains global simulation parameters. In this case, we have specified the output directory to be `output_tutorial_0_minimal`.

The simulator queries the `PopulationConfigGenerator` each generation to obtain a population configuration, which contains the information necessary to create the next generation from the previous one. In this case, the population configuration is the same each generation — it tells the simulator to create a population of 10 individuals with parents drawn from the same population in the previous generation.

When you run `forqs` on this example (`tutorial_0_minimal.txt`), you will see screen output showing that the simulation ran for 3 generations. You will also see the new output directory `output_tutorial_0_minimal`. If you look in this directory, you'll see a single file: `forqs.simconfig.txt`. This file contains the configuration used for the simulation, including unspecified default parameters. However, there are no other output files in the directory, so we don't actually know what happened during the simulation. To get more information, we need to specify Reporters to report output — we show how to do this in the next section.

3.6.1 Recombination and reporting output

In this example, we add modules to include recombination and report output:

```
#
# tutorial_1_recombination_reporter.txt
#

PopulationConfigGenerator_ConstantSize pcg
    generation_count = 3
    population_count = 1
    population_size = 10
    chromosome_lengths = 1e6
```

```
RecombinationPositionGenerator_Uniform rpg
    rate = 1

Reporter_Population reporter_population
    # update_step = 1

SimulatorConfig
    output_directory = output_tutorial_1_recombination_reporter
    population_config_generator = pcg
    recombination_position_generator = rpg
    reporter = reporter_population
```

When creating offspring chromosomes from parental chromosomes, the simulator uses the `RecombinationPositionGenerator` module to generate lists of recombination positions. In this case, we are using `RecombinationPositionGenerator_Uniform`, which chooses the positions uniformly at random along the chromosome. Note the use of the compound parameter `chromosome_length_rate` that specifies both the length of the chromosome and the recombination rate.

`Reporter` modules are used by `forqs` to output information about the simulation. The user can specify an arbitrary number of `Reporters`, depending on what output is desired. `Reporter_Population` outputs `forqs` population files, where each individual is represented as a list of chromosomes (one chromosome per line) and each chromosome is represented as a list of haplotype chunks. By default, population files will be produced only for the final generation — setting `update_step = n` will tell the `Reporter_Population` to output population files every `n` generations. You can see this by uncommenting (removing the '#' character) the `update_step` line in the configuration file and re-running the simulation.

The population files contain the raw information about the simulated individuals, and are not meant to be analyzed directly. Instead, they can be used to propagate existing neutral variation from the founding haplotypes of the initial generation to the mosaic haplotypes of the final generation (see Section 3.8).

However, we will use the population files to illustrate `forqs`' internal representation of chromosomes. The first chromosome in the population file from the final generation is:

```
+ { (0,12) (704958,7) }
```

This means that the first chromosome of the first individual is made up of two haplotype chunks: the first chunk (0,12) starts at position 0 and comes from founder individual 12; the second chunk (704958,7) starts at position 704958 and comes from founder individual 7. Now comment out the following line in `SimulatorConfig`:

```
recombination_position_generator = rpg
```

which 'un-plugs' the `RecombinationPositionGenerator` from the simulator. Now re-run the simulation, specifying a different output directory on the command line:

```
forqs tutorial_1_recombination_reporter.txt output_directory=out2
```

When you look at the resulting final population file, you will see that all chromosomes consist of a single chunk, as you would expect with no recombination.

3.6.2 Wright-Fisher simulation

Our next example is a simple Wright-Fisher simulation where we track the allele frequency at a single locus under neutral drift:

```
#
# tutorial_2_wright_fisher.txt
#

PopulationConfigGenerator_ConstantSize pcg
    population_size = 100
    generation_count = 200
    # population_count = 10
```

```
Locus my_locus
    chromosome = 1
    position = 100000

VariantIndicator_SingleLocusHardyWeinberg vi
    locus = my_locus
    allele_frequency = .5

Reporter_AlleleFrequencies reporter_allele_freqs
    locus = my_locus

SimulatorConfig
    output_directory = output_tutorial_2_wright_fisher
    population_config_generator = pcg
    variant_indicator = vi
    reporter = reporter_allele_freqs
```

In this example, the population size is 100, the initial allele frequency is .5, and the simulation runs for 200 generations.

The `Locus` module defines a single site — in this case it is position 100000 on chromosome 1. The `Locus` has id `my_locus`, and this id is used by two other modules to refer to this locus.

As we saw in the previous example, `forqs` represents chromosomes as lists of haplotype chunks, with no information about nucleotides or variant (SNP) values at any position. (To be concrete, we think of a variant value as being 0 or 1, but `forqs` variant values can be anything in the range 0–255, so we are not restricted to bi-allelic SNPs). However, if we give `forqs` the variant values for the founding haplotypes, it can calculate variant values for any mosaic chromosome. This information is represented by a `VariantIndicator`, which is a function that tells `forqs` which founding chromosomes contain which variants.

In our case, we specify a `VariantIndicator_SingleLocusHardyWeinberg`, which assigns variants to individuals in Hardy-Weinberg proportions. Because we have specified an initial allele frequency of .5, this means that of the founding individuals, 25% will be homozygote 0, 25% will be homozygote 1, and 50% will be heterozygotes.

We also want to track the allele frequency at this locus, so we specify a `Reporter_AlleleFrequencies`, which reports the allele frequency at each generation. You will find the output in the file `allele_frequencies_chr1_pos100000.txt`.

You may have noticed the commented line with the parameter `population_count = 10`. By uncommenting this line, you will tell `forqs` to simulate 10 populations. `PopulationConfigGenerator_ConstantSize` does not allow migration between populations, so this effectively gives 10 independent Wright-Fisher simulations. The resulting allele frequency trajectories can all be found in the same output file, with one column per population. As expected, you will see that the alternate allele (1) has been fixed or lost in some populations, and is still polymorphic in others.

3.6.3 Selection

In this example, we add selection to our Wright-Fisher simulation.

```
#
# tutorial_3_selection.txt
#

PopulationConfigGenerator_ConstantSize pcg
    population_size = 100
    generation_count = 100
    population_count = 10
    fitness_function = qt

Locus my_locus
    chromosome = 1
    position = 100000

VariantIndicator_SingleLocusHardyWeinberg vi
    locus = my_locus
    allele_frequency = .5

QuantitativeTrait_SingleLocusFitness qt
```

```
    locus = my_locus
    w0 = 1
    w1 = 1.1
    w2 = 1.2

Reporter_AlleleFrequencies reporter_allele_freqs
    locus = my_locus

Reporter_TraitValues reporter_trait_values
    quantitative_traits = qt

Reporter_DeterministicTrajectories reporter_deterministic_trajectories
    initial_allele_frequency = .5
    w0 = 1
    w1 = 1.1
    w2 = 1.2

SimulatorConfig
    output_directory = output_tutorial_3_selection
    population_config_generator = pcg
    variant_indicator = vi
    quantitative_trait = qt
    reporter = reporter_allele_freqs
    reporter = reporter_trait_values
    reporter = reporter_deterministic_trajectories
```

For this simulation, we add a quantitative trait representing fitness determined by an individual's genotype at a single locus: `QuantitativeTrait_SingleLocusFitness`. The parameters `w0, w1, w2` are used to specify the fitness for individuals with genotype 0, 1, and 2, respectively. In general, a quantitative trait can depend on multiple loci on multiple chromosomes, as well as on other quantitative traits. Note that the name of this quantitative trait `qt` is passed to the `PopulationConfigGenerator` as the `fitness_function` parameter.

We also add some useful `Reporters` that output the mean fitnesses of the populations as well as the deterministic trajectories expected in the limit of infinite population size. These output files can be easily read into a plotting application (e.g. `R`) to compare the random trajectories

with the deterministic trajectories (Figure 3.4).

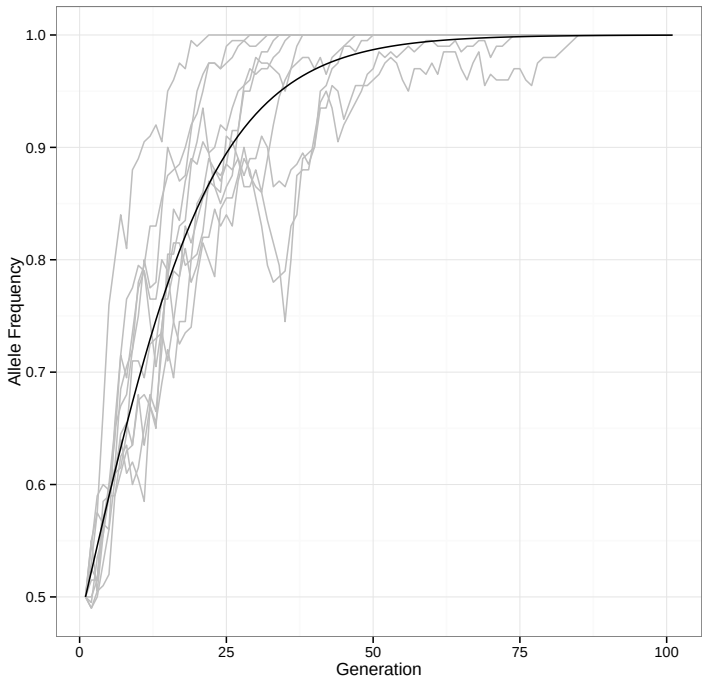

Figure 3.4: **Allele frequency trajectories from selection simulation**.

3.6.4 Trajectories

This section shows how to use `Trajectory` modules to specify population sizes that change over time. In general, `Trajectory` modules can be used to specify any numeric value that varies by population and generation – other examples include mutation rates and migration rates.

In the first example (`tutorial_4a_trajectories.txt`), there are two populations following the same population size trajectory: constant size 100 until generation 2, followed by a linear increase in population size until it reaches 400 at generation 5, after which the population size stays at 400. This is accomplished by specifying each piece (`Trajectory_Constant`, `Trajectory_Linear`, `Trajectory_Constant`) and composing the pieces together with

Trajectory_GenerationComposite.

```
#
# tutorial_4a_trajectories.txt
#

Trajectory_Constant popsize_1
    value = 100

Trajectory_Linear popsize_2
    begin:value = 2 100
    end:value = 5 400

Trajectory_Constant popsize_3
    value = 400

Trajectory_GenerationComposite popsize
    generation:trajectory = 0 popsize_1
    generation:trajectory = 2 popsize_2
    generation:trajectory = 5 popsize_3

PopulationConfigGenerator_LinearSteppingStone pcg
    generation_count = 8
    population_count = 2
    population_size = popsize

SimulatorConfig
    output_directory = output_tutorial_4a_trajectories
    write_popconfig = 1        # set this flag to write out the popconfig file
    population_config_generator = pcg
```

Note that PopulationConfigGenerator_LinearSteppingStone has a population_size parameter whose value must be the id of a Trajectory module (rather than a numeric value). Also, unless otherwise specified, each Trajectory gives the same numeric value for each population in a given generation. You can see the raw population configuration generated by including the parameter write_popconfig = 1, which tells

`forqs` to write out the population configs in a file `forqs.popconfig.txt`. After running this example, you can examine `forqs.popconfig.txt` to see the population sizes.

In the next example (`tutorial_4b_trajectories.txt`), population 1 follows the same population size trajectory as in the first example, while population 2 experiences exponential growth, doubling each generation (with initial population size 1000). In order to specify this scenario, each population size trajectory is specified separately (`popsize_pop1` and `popsize_pop2` in this example), and the two are composed with `Trajectory_PopulationComposite`. When `Trajectory_PopulationComposite` is queried for a value for a population, it returns the value obtained from the appropriate sub-trajectory.

```
#
# tutorial_4b_trajectories.txt
#

# population 1

Trajectory_Constant popsize_pop1_1
    value = 100

Trajectory_Linear popsize_pop1_2
    begin:value = 2 100
    end:value = 5 400

Trajectory_Constant popsize_pop1_3
    value = 400

Trajectory_GenerationComposite popsize_pop1
    generation:trajectory = 0 popsize_pop1_1
    generation:trajectory = 2 popsize_pop1_2
    generation:trajectory = 5 popsize_pop1_3

# population 2

Trajectory_Exponential popsize_pop2
    generation_begin = 0
```

```
    value_begin = 1000
    rate = 0.693147     # log(2); population size doubles each generation

# composite trajectory

Trajectory_PopulationComposite popsize
    trajectories = popsize_pop1 popsize_pop2

PopulationConfigGenerator_LinearSteppingStone pcg
    generation_count = 8
    population_count = 2
    population_size = popsize

SimulatorConfig
    output_directory = output_tutorial_4b_trajectories
    write_popconfig = 1        # set this flag to write out the popconfig file
    population_config_generator = pcg
```

3.6.5 Quantitative traits

In this section, we present an example of specifying a quantitative trait by specifying quantitative trait loci (QTLs) and their effect sizes.

```
#
# tutorial_5_qtl.txt
#

PopulationConfigGenerator_ConstantSize pcg
    generation_count = 10
    population_count = 1
    population_size = 100
    chromosome_pair_count = 3
    chromosome_lengths = 1e6 1e6 1e6
    fitness_function = ff

Locus locus1
    chromosome = 1
    position = 1000
```

```
Locus locus2
    chromosome = 2
    position = 2000

Locus locus3
    chromosome = 3
    position = 3000

LocusList loci
    loci = locus1 locus2 locus3

VariantIndicator_Random vi
    locus_list:population:frequencies = loci * .5 .5 .5

QuantitativeTrait_IndependentLoci qt
    qtl = locus1 0 .1 .2
    qtl = locus2 0 .2 .4
    qtl = locus3 0 .1 .2
    environmental_variance = .05

FitnessFunction_TruncationSelection ff
    quantitative_trait = qt
    proportion_selected = .5

Reporter_AlleleFrequencies reporter_allele_frequencies
    quantitative_trait = qt

Reporter_TraitValues reporter_trait_values
    quantitative_traits = qt
    write_full = 1

SimulatorConfig
    output_directory = output_tutorial_5_qtl
    population_config_generator = pcg
    variant_indicator = vi
    quantitative_trait = qt
    quantitative_trait = ff
    reporter = reporter_allele_frequencies
    reporter = reporter_trait_values
```

```
write_vi = 1
```

In this example, we specify a single population (size 100) where individuals have 3 chromosomes. We specify 3 loci, each on a different chromosome.

`VariantIndicator_Random` is used to assign variants to the initial haplotypes randomly, according to the specified allele frequencies – in this case, the variant allele frequency is .5 for each locus. The `'*'` indicates that all populations have the same allele frequencies – because there is only one population, we could have used `'1'` in place of `'*'`. The parameter `write_vi = 1` in `SimulatorConfig` tells `forqs` to write the file `forqs.vi.txt` containing lists of the initial haplotypes carrying the variant allele at each locus.

`QuantitativeTrait_IndependentLoci` specifies a quantitative trait where the effects of variants at different loci are independent of each other, i.e. there are no epistatic effects. Each QTL is listed with its per-genotype effects – in this case, individuals with genotype 0, 1, or 2 at `locus1` will have 0, .1, or .2 added to their trait value, respectively. Alternatively, QTL effect sizes and dominance values can be chosen randomly according to specified distributions. For an example of this, see `example_qtl_random.txt` which also demonstrates the use of `LocusList_Random` to pick the loci randomly as well.

In this example, `FitnessFunction_TruncationSelection` assigns fitness values of 1 to individuals in the upper half of the distribution, and 0 to individuals in the lower half (`proportion_selected = .5`). `Reporter_AlleleFrequencies` uses the parameter `quantitative_trait = qt`, which tells it to report allele frequencies for all QTLs associated with the trait. `Reporter_TraitValues` reports mean trait values for the population each generation; the parameter `write_full = 1` tells it to also report individual trait values for each generation.

3.6.6 Reporting haplotype frequencies

This example shows the use of `Reporter_HaplotypeFrequencies` to report local haplotype frequencies in the simulated populations.

In this example, there is a single locus that confers an additive fitness advantage to the individuals carrying the selected variant. The `VariantIndicator` specifies which haplotype ids actually carry the variant: in this case, `VariantIndicator_IDRange` assigns the value `1` to the first 1000 haplotype ids, i.e. the first 500 individuals are homozygous for the selected variant.

`Reporter_HaplotypeFrequencies` uses a `HaplotypeGrouping` module that specifies how haplotype ids should be grouped. The grouping can be as fine-grained as a single haplotype id per group, but for simplicity we have specified just two groups: those individuals who carry the selected variant, and those who don't. `Reporter_HaplotypeFrequencies` then reports local proportions of the two groups in the population.

We have specified a relatively high recombination rate (3 crossovers per meiosis on average) and strong selection (.1 selection coefficient) in order to illustrate the change in local haplotype frequencies in just a few generations. After running the example, the output file `haplotype_frequencies_chr1_final_pop1.txt` shows that haplotypes carrying the selected variant have risen to high frequency near the selected locus, but not as much in regions farther away, due to recombination. The parameter `update_step = k` will result in reporting of local haplotype frequencies every `k` generations (default is to report only the final generation).

```
#
# tutorial_6_haplotype_frequencies.txt
#

PopulationConfigGenerator_ConstantSize pcg
    generation_count = 20
    population_count = 1
```

```
    chromosome_pair_count = 1
    chromosome_lengths = 1e6
    population_size = 1000
    fitness_function = qt

RecombinationPositionGenerator_Uniform rpg
    rate = 3

Locus selected_locus
    chromosome = 1
    position = 500000

VariantIndicator_IDRange vi
    locus:start:count:step:value = selected_locus 0 1000 1 1

QuantitativeTrait_IndependentLoci qt
    qtl = selected_locus 1 1.1 1.2

Reporter_AlleleFrequencies reporter_allele_frequencies
    quantitative_trait = qt

HaplotypeGrouping_Uniform hg
    ids_per_group = 1000        # 2 groups of 500 individuals

Reporter_HaplotypeFrequencies reporter_haplotype_frequencies
    haplotype_grouping = hg
    chromosome_step = 1e5
    #update_step = 1

SimulatorConfig
    output_directory = output_tutorial_6_haplotype_frequencies
    population_config_generator = pcg
    recombination_position_generator = rpg
    variant_indicator = vi
    quantitative_trait = qt
    reporter = reporter_allele_frequencies
    reporter = reporter_haplotype_frequencies
```

3.7 Modules

In this section, we give an overview of the `forqs` module interfaces. For information about specific `forqs` module implementations, including parameter names, usage, and links to examples, please refer to the `forqs` Module Reference (`forqs_module_reference.html`).

3.7.1 Module types and interfaces

`forqs` has a single *top-level module* called `SimulatorConfig`. There are seven *primary modules*, corresponding to the orange boxes Figure 3.3, which represent the main places where `forqs` can be customized. There are also several *building block modules*, which are used by the primary modules.

Multiple modules may share a common interface, as indicated by their names. For example, `Reporter_Timer` and `Reporter_AlleleFrequencies` both implement the `Reporter` interface. Modules with the same interface can be used interchangeably by the simulator.

In addition, specific modules can be instantiated multiple times (with different object ids and parameters) – for example, `Reporter_AlleleFrequencies` can be used to report allele frequencies at several different loci.

By mixing and matching different modules, the user has a great deal of flexibility in creating customized simulation scenarios. In addition, `forqs` can easily be extended with new functionality by adding new modules that implement existing module interfaces.

3.7.2 Module specification and instantiation

Each `forqs` module specified in the configuration file corresponds to an object that is instantiated by the simulator before the simulation starts. The first line of a module specification contains the module name and object id. Each subsequent line specifies a parameter of the mod-

ule, formatted as a name-value pair: *name* = *value*. The value is read as a string, but may be interpreted as an integer, floating point, string, or list, depending on the parameter. Each module determines how its parameter values are interpreted, and interpretation errors are reported to the user. In some modules, multiple parameter values may be specified with the same name, in which case each value specified is appended to a list. A blank line ends the specification for that module.

As an example, here is the specification of a `Locus` module, which represents a single nucleotide position:

```
Locus my_locus
    chromosome = 1
    position = 500000
```

The first line specifies a `Locus` object with id `my_locus`. The two parameters specify which chromosome (1) and the position on the chromosome (500000).

After parsing this module specification, the simulator instantiates a `Locus` object, and puts the object in a registry under the name `my_locus`. Subsequently instantiated objects may need a reference to this object, which can be obtained by looking up the object by name in the registry. Because modules are instantiated in the order they are specified in the configuration file, it is important to specify an object before it is referenced by other objects (otherwise `forqs` will produce an error message that it couldn't find the referenced object in the registry).

Object references are specified as string-valued parameters, where the value contains the object id of the referenced object. For example, the following is the specification of a `Reporter` module that holds a reference to our example `Locus` object `my_locus`:

```
Reporter_AlleleFrequencies reporter_allele_frequencies
    locus = my_locus
```

One final note on a parameter naming convention used in `forqs` – for parameters where the value is a list of sub-parameters, the parameter name should be a concatenation of the sub-parameters. This convention allows easier reading and editing of configuration files without the need to consult documentation. For example, `Trajectory_GenerationComposite` represents a list of `Trajectory` modules, to be followed piecewise at specified generations.

This module is parametrized by a list of pairs (generation, trajectory), using the parameter name `generation_trajectory`:

```
Trajectory_GenerationComposite id_migration_rate
    generation_trajectory = 0 id_migration_rate_0
    generation_trajectory = 5 id_migration_rate_1
```

In this case, the module configuration can be interpreted as follows: at generation 0 use trajectory `id_migration_rate_0`, and then at generation 5 start using trajectory `id_migration_rate_1`.

3.7.3 Top-level module: SimulatorConfig

`SimulatorConfig` is the top-level module containing:

- global simulation parameters
- references to the primary modules

Because `SimulatorConfig` has references to the primary modules, it must be specified last in the configuration file.

3.7.3.1 Global simulation parameters

The main global simulation parameters are:

- `output_directory`: `forqs` will create this directory and place all output files here
- `seed`: seed for the random number generator

Command line parameters can also be specified on the command line as *name=value* with no whitespace, as shown in this example:

```
forqs config.txt output_directory=mydir seed=12345
```

If a parameter is specified on both the command line and in the configuration file, the command line takes precedence.

If no random seed is specified, `forqs` will first look in the current working directory for the file `forqs.seed` containing a previously generated seed. If no seed is found, `forqs` will generate a new seed from the system time. At the end of each run, `forqs` generates a random seed and writes it to `forqs.seed`. To summarize, `forqs` looks for the seed in the following places, in order of preference:

1. command line
2. configuration file
3. `forqs.seed`
4. system time

3.7.3.2 References to primary modules

Of the primary modules, only `PopulationConfigGenerator` is required to be specified, because it contains necessary information about population sizes and the number of generations. The rest of the modules default to trivial implementations. The primary modules specified in `SimulatorConfig` are:

1. `PopulationConfigGenerator`
    - required – no default
2. `RecombinationPositionGenerator`
    - default: `RecombinationPositionGenerator_Trivial` (no recombination – whole chromosomes are transmitted, chromosome pairs segregate independently)

3. `MutationGenerator`

    - default: none (no mutation)

4. `VariantIndicator`

    - default: `VariantIndicator_Trivial` (always returns 0)

5. `QuantitativeTrait`

    - default: none (no quantitative traits)
    - multiple `QuantitativeTraits` may be defined

6. `Reporter`

    - default: none (no reporters)
    - multiple `Reporters` may be defined

3.7.4 Primary modules

3.7.4.1 PopulationConfigGenerator

For the reproduction/transmission step, in addition to information about the current populations, the simulator needs to know the *population configuration* for the next generation, which includes:

- number and sizes of the populations in the next generation
- mating distribution, which describes how parents are chosen from populations in the current generation to create offspring in the next generation, including which quantitative traits to use as fitness functions

The PopulationConfigGenerator module provides an interface for the simulator to obtain a population configuration for each generation. To create each new individual in the next generation, the simulator chooses parent populations according to the mating distribution, and then individual parents with probability proportional to their fitnesses.

For the initial generation, individuals are assigned haplotype ids sequentially starting at 0, with two ids assigned per individual (one for the maternal chromosomes, and one for the paternal chromosomes). Id offsets can be used to make it easier to distinguish between populations – for example, individuals from population 1 may be assigned ids starting at 0, and individuals from population 2 may be assigned ids starting at 1000000.

PopulationConfigGenerator_File allows the user to specify a population configuration for each generation, giving the user precise control over the demographic histories of the simulated populations.

Alternatively, forqs provides higher level PopulationConfigGenerators that allow the user to specify time-dependent population size and migration rate trajectories (e.g. PopulationConfigGenerator_LinearSteppingStone). When a migration rate is specified from a source population to a destination population, it is interpreted to be the probability that a new child in the destination population has parents in the source population. Equivalently, it is the expected proportion of individuals in the next generation of the destination population whose parents were in the source population.

For debugging purposes, forqs optionally outputs the file forqs.popconfig.txt with the population configuration that was used for each generation during the simulation. This option can be selected by setting the write_popconfig parameter in SimulatorConfig:

```
write_popconfig = 1
```

3.7.4.2 RecombinationPositionGenerator

The `RecombinationPositionGenerator` module provides an interface for the simulator to obtain a list of recombination positions. After querying the `RecombinationPositionGenerator`, the simulator uses the list of recombination positions to create an offspring chromosome from a pair of parental chromosomes.

There are multiple implementations of the `RecombinationPositionGenerator` module interface, each of which uses a different method to generate the recombination position list.

`RecombinationPositionGenerator_SingleCrossover` assumes there is a single crossover event at a position chosen uniformly at random over that chromosome. This results in the offspring receiving one of the two parental or one of the two recombined chromosomes with equal 25% probabilities.

`RecombinationPositionGenerator_Uniform` generates recombination positions according to a uniform Poisson process along the chromosome, with rate specified by the user.

`RecombinationPositionGenerator_RecombinationMap` generates recombination positions according to previously estimated recombination maps, in the format used by the HapMap project.

3.7.4.3 MutationGenerator

The user may specify a `MutationGenerator` module to generate random mutations during the simulation. `MutationGenerator` provides a single interface through which the simulator obtains a list of new mutations for a particular generation. However, `MutationGenerator` implementations may differ in how they generate this list of mutations. For example, `MutationGenerator_SingleLocus` generates mutations only at a single site, while `MutationGenerator_Regions` generates mutations in multiple regions

with different (possibly time-dependent) mutation rates. In any case, the user will most likely want to also specify `Reporter_Regions` to report the final mutated sequences at the end of the simulation.

`forqs` current mutation implementation creates a new haplotype id for each new mutation, storing it in a haplotype ancestry tree (so that information about the ancestral haplotype can be preserved). This results in memory usage that increases with each generation, which may cause performance degradation in simulations involving a high total mutation rate for a large number of generations. (Here *total mutation rate* means θL, where $\theta = 4N\mu$ is the population-scaled per-site mutation rate and L is the total length of the genomic region where new mutations are being generated).

3.7.4.4 VariantIndicator

Internally, `forqs` represents each individual chromosome as a list of haplotype chunk ids, with no information about particular variants carried on that chromosome. In order to obtain an individual's genotype at a locus, the simulator must use a `VariantIndicator`, which is essentially a function that maps:

$$(\textit{locus}, \textit{haplotype id}) \mapsto \textit{variant value}$$

Typically these variant values will be 0/1 for SNPs; however, any value in the range 0-255 can be used.

In general, a `VariantIndicator` needs to specify variant values only at selected loci, because neutral loci have no effect on the simulation. Ancestral neutral variation that is present on the haplotypes of the founders (individuals in the initial generation) can be propagated to individuals at the end of the simulation using the `forqs_map_ms` tool included in the `forqs` package.

Several `VariantIndicator` implementations are available, allowing the user a variety

of options for specifying the variants carried by the founders – these can be found in the `forqs` Module Reference.

3.7.4.5 QuantitativeTrait

A `QuantitativeTrait` module represents a single quantitative trait. The module encapsulates the information needed to calculate the trait value for each individual based on the individual's genotypes at a list of user-specified loci. For example, `QuantitativeTrait_IndependentLoci` allows the user to specify loci and effect sizes, and the trait value for each individual is calculated by combining that individual's locus-specific effects additively (with user-specified environmental variance).

In addition, a quantitative trait may depend on other quantitative traits. `QuantitativeTrait_Expression` allows the user to define a quantitative trait by a mathematical expression whose variables are other quantitative traits.

Some `QuantitativeTraits` are named with the prefix `FitnessFunction`, which indicates that they are intended to be used at fitness functions for choosing individuals. However, any `QuantitativeTrait` may be used as a fitness function. Full documentation on the available `QuantitativeTraits` can be found in the `forqs` Module Reference.

3.7.4.6 Reporter

`Reporter` modules are used to report information during the simulation. For example, `Reporter_AlleleFrequencies` reports the allele frequency of a user-specified locus at each generation. The full list of available `Reporters` can be found in the `forqs` Module Reference.

3.7.5 Building block modules

3.7.5.1 Locus and LocusList

The `Locus` module represents a single site on a chromosome, specified by the chromosome pair index and position on the chromosome:

```
Locus my_locus
    chromosome_pair_index = 0
    position = 500000
```

Note that the chromosome pairs use 0-based indexing, so `chromosome_pair_index = 0` refers to the first chromosome pair.

`LocusList` modules are used to define a list of loci. The list can be specified in two ways – in the first, loci are specified by chromosome number and position:

```
LocusList locus_list
    chromosome:position = 1 1000123
    chromosome:position = 2 2000234
    [...]
```

Alternatively, the loci can be references to previously specified `Locus` objects:

```
LocusList locus_list_2
    loci = my_locus_1 my_locus_2 [...]
```

Also, a random list of loci can be generated with `LocusList_Random`:

```
LocusList_Random locus_list_random
    locus_count = 10
```

3.7.5.2 Trajectory

In simple situations, constant parameter values can be used to specify an aspect of the simulation. For example, it is common in population genetics to simulate models where population sizes are constant. In more complicated scenarios, parameter values (e.g. population sizes, migration

rates, mutation rates, optimal quantitative trait values) may vary in space (population) or in time (generation).

`forqs` uses `Trajectory` modules to describe such varying quantities. In essence, a `Trajectory` represents a function:

$$(\textit{population index}, \textit{generation index}) \mapsto \textit{value}$$

Trajectories can be built by the user by starting with simple building blocks (e.g. constant, linear, polynomial, exponential functions) and then composed either by population or by generation. Once the `Trajectory` has been defined, it can be referenced by name to be used as the parameter of another module. Note that if a module expects a particular parameter to be a `Trajectory`, it is an error to pass a constant integer or floating point number as the parameter value. In this case, if the user really wants a constant parameter value, `Trajectory_Constant` should be specified.

3.7.5.3 Distribution

`Distribution` modules represent probability distributions. These modules are used to model, for example, effect sizes for QTLs or environmental variance for a quantitative trait.

`Distribution_Constant` can be used in situations where a module requires a reference to a `Distribution` but a constant value is desired.

3.8 Simulating background variation

Forward-in-time simulators often use a burn-in period to allow neutral variation to reach mutation-drift equilibrium. An alternative strategy, adopted by `forqs`, is to use an existing program (such as Dick Hudson's `ms`) to generate neutral variation for the founders in the initial population. Chromosomes in the final populations are mosaics of the founder chromosomes, so neutral variation can be propagated to the mosaic chromosomes to generate sequence (or SNP) data. The tool `forqs_map_ms` is included in `forqs` packages for this purpose. In this section we discuss the propagation procedure using two examples.

3.8.1 Example 1: No new mutations

The first example is a simple case where there are no new mutations introduced during the `forqs` simulation. This case includes many scenarios of interest where the variation in the population has been generated over a small number of generations, primarily by recombination and recent admixture between historically isolated (or inbred) populations.

In this case, the propagation of neutral variation is straight-forward: sequence variants on founder haplotypes are mapped onto the mosaic chromosomes of the final population. However, there are many details about the mapping that must be specified by the user in a mapping configuration file. `forqs` produces final populations of individuals that may carry multiple pairs of chromosomes, with absolute positions specified by integers. `ms` outputs single chromosomes, with relative positions specified by floating point numbers. The mapping configuration thus needs to specify an integer range of positions on a particular chromosome that correspond to the `ms` relative positions in the range [0,1]. In addition, the user must specify how `forqs` haplotype ids correspond to the `ms`-format variant sequences.

In the `examples` directory, the following 4 files can be found:

- `population_example.txt` (`forqs` population data)

- `ms_test_data_1.txt, ms_test_data_2.txt` (`ms`-format files)
- `ms_map_config.txt` (mapping configuration file)

Perform the mapping using the following command:

```
forqs_map_ms population_example.txt ms_map_config.txt > output.ms
```

The resulting sequences in `output.ms` are mosaics of the sequences in the input `ms`-format files. For example, the first chromosome of the first individual is a mosaic of haplotypes 0, 1, 2, and 3, corresponding to the sequences 'aaaaaaaaa', 'bbbbbbbbb', 'ccccccccc', and 'ddddddddd', respectively. The output sequence for this chromosome is 'aabccccdd'. Note that alphabetic letters are used in these example sequences for illustration only – real `ms` output consists of 0's and 1's.

3.8.2 Example 2: Including new mutations

In scenarios where it is important to include new mutations in addition to ancestral neutral variation, the mapping procedure has one additional complication. When simulating forward in time, the neutral variants carried by individuals have no effect on the dynamics of the simulation. `forqs` takes advantage of this by ignoring (i.e. not storing in memory) any ancestral neutral variation on the founding haplotypes. Each new mutation generated in `forqs` results in a new haplotype id. Because of this, it is necessary to use an id ancestry map to translate the new ids back to the ancestral ids before performing the mapping. After the neutral variation has been mapped, the new mutations can then be merged with the ancestral variants. This procedure is detailed in the following shell script (and data files referenced in the script), which runs a complete example of a `forqs` simulation followed by mapping `ms` sequences (with new mutations merged):

```
examples/ms_map_example_new_mutations.sh
```

3.9 Validation

`forqs` has an extensive set of unit tests that verify the correctness of individual code modules. In addition, in order to validate the larger-scale behavior of the simulations, we have compared `forqs` simulation results to theoretical predictions from population genetics and quantitative genetics.

3.9.1 Single locus selection

We first considered simple scenarios where an individual's fitness is determined by that individual's genotype at a single locus, under a wide range of fitness effect sizes and dominance values. We compared the simulated allele frequency trajectories to the deterministic trajectories predicted in the limit of infinite population size.

In all cases, the simulated trajectories closely followed the deterministic trajectories, with better agreement in simulations with larger population sizes, as expected. Figure 3.5 shows the case of additive positive selection, for population sizes of 100 and 1000.

3.9.2 Decay of linkage disequilibrium

In the limit of infinite population size, linkage disequilibrium (measured by D, the correlation between two variants at different sites) decays geometrically at rate $1 - r$, where r is the recombination rate (see Figure 3.6 for an example run).

3.9.3 Mutation-drift equilibrium

To validate our mutation implementation, we approximated the infinite-sites model by simulating mutations generated randomly in large regions (1mb). We set the per-site mutation rate to make the population-scaled mutation rate $\theta = 10$, and simulated for enough generations to reach

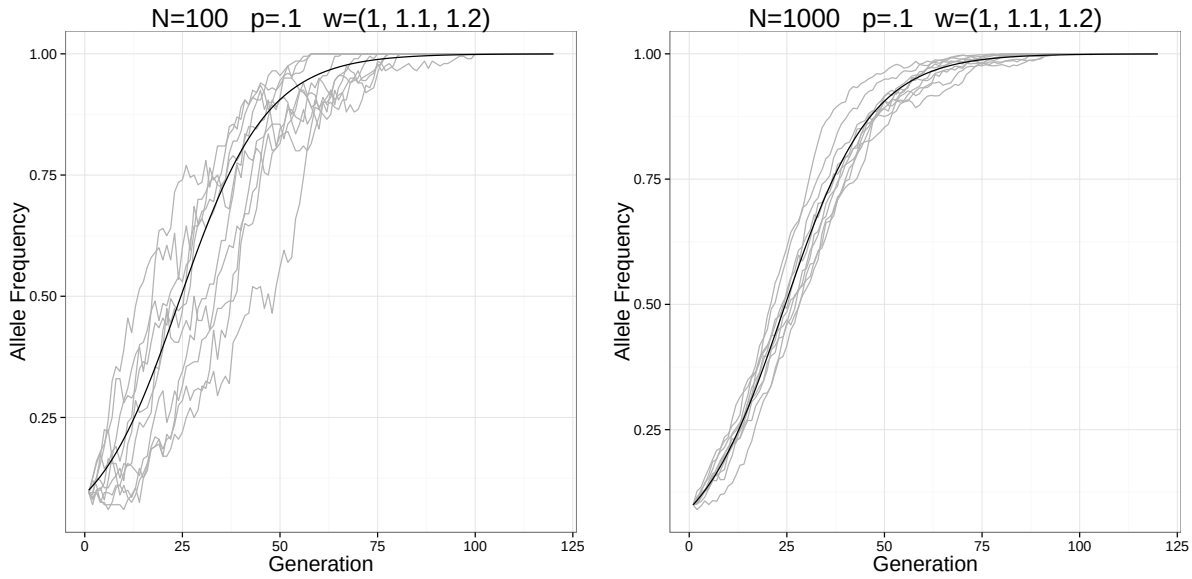

Figure 3.5: **Selection on a single locus**. The dark curves show the deterministic trajectories, and the lighter curves show simulated trajectories. Populations of size 100 and 1000 were simulated with additive fitness effect .1 at a single locus, with the selected variant having initial allele frequency .1.

mutation-drift equilibrium.

We then compared the resulting site-frequency spectra $\{\xi_i\}$ to theoretical predictions, where ξ_1 is the number of singletons, ξ_2 is the number of doubletons, etc. in the sample. Coalescent theory predicts that $\xi_i = \theta/i$ in expectation (Fu, 1995), under the assumption that the sample size is small compared to the population size. When the sample size is close to the population size, the Wright-Fisher model is expected to have approximately 12% more singletons and 2% less doubletons compared to the coalescent expected values (Wakeley and Takahashi, 2003).

Site frequency spectra generated from the full population agree with the Wright-Fisher large sample expected values, and site frequency spectra from smaller samples agree with the coalescent expected values, as expected (Figure 3.7).

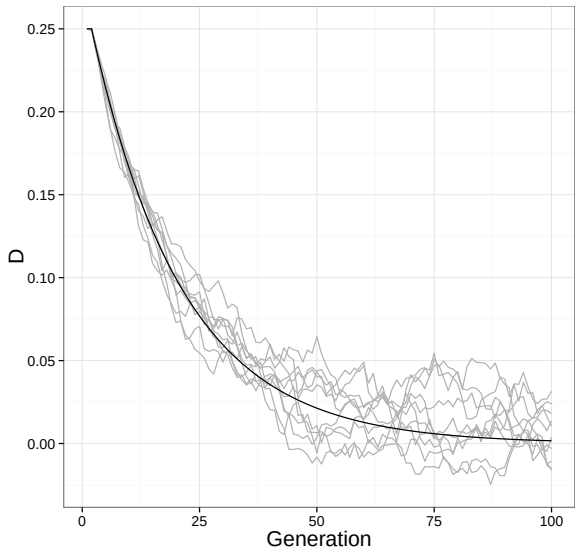

Figure 3.6: **Decay of linkage disequilibrium**. The dark curve shows the deterministic trajectory, and the lighter curves show simulated trajectories. In this example, $D = .25$ initially and $r = .05$.

3.9.4 Response to selection

To validate our implementation of selection on quantitative traits, we simulated a quantitative trait with 10 QTLs with identical additive effect sizes and initial allele frequencies, which gives an approximately normal distribution of trait values in the population. We also used the `forqs` quantitative trait `FitnessFunction_TruncationSelection`, which selects a specified proportion of individuals at the upper tail of the trait value distribution to produce offspring for the next generation.

The Breeder's Equation from quantitative genetics (Gillespie, 2004; Falconer and Mackay, 1996) predicts the response to selection R from the heritability of the trait h^2 and the selection

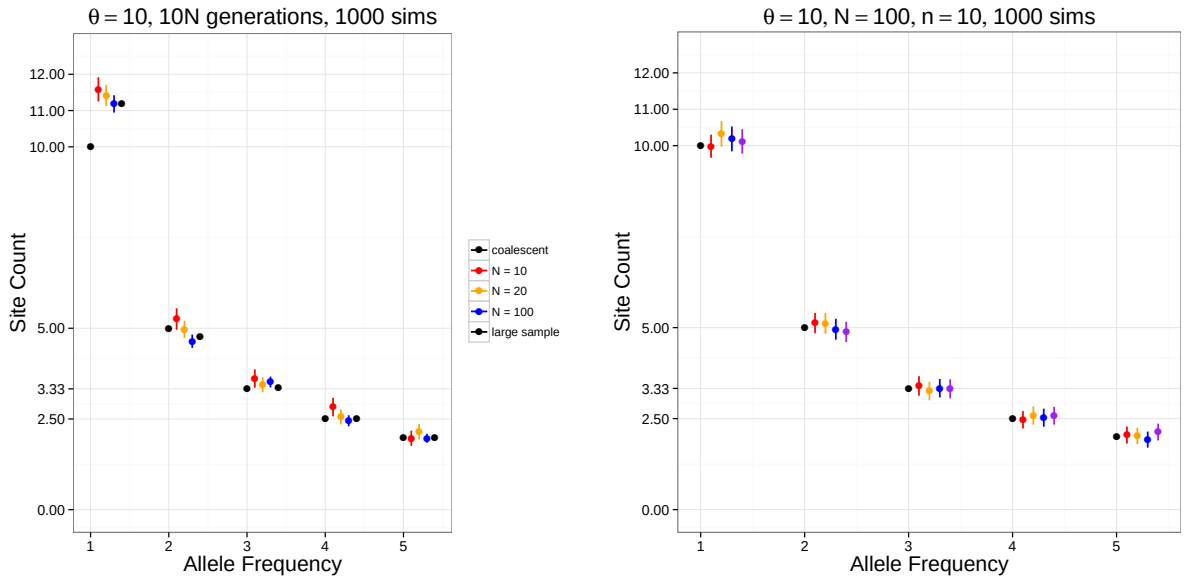

Figure 3.7: **Mutation-drift equilibrium.** (*left*) Simulations for different population sizes are shown, together with expected values for both the coalescent and Wright-Fisher with large sample size. (*right*) Small samples taken from a larger simulated population agree with the coalescent expected values.

differential S:

$$R = h^2 S$$

where the S and R are the selected parent mean and offspring mean, respectively, measured as deviations from the population mean.

Simulations run with various values for the heritability and proportion of individuals selected agree with the response values predicted from the Breeder's Equation (Figure 3.8).

3.9.5 Performance

The representation of chromosomes as haplotype chunks in `forqs` makes very efficient use of memory, independent of the size of the chromosomes. This allows, for example, simulation

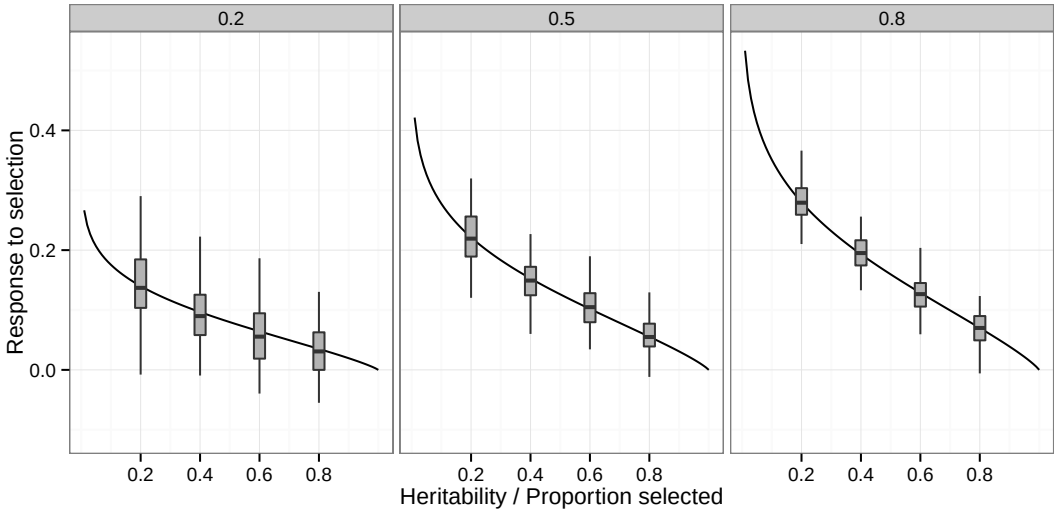

Figure 3.8: **Response to selection.** 3 panels show simulations run at different heritability levels (.2, .5, .8), with varying proportions of selected individuals (.2, .4, .6, .8) (100 simulations each). The dark curves show the response values predicted from the Breeder's Equation.

of entire genomes of individuals. One drawback to this approach is that memory usage grows linearly with the number of generations, due to recombination.

On a typical laptop computer, for a population size of 1 million, simulations take ~ 1.5 seconds per generation for neutral scenarios and ~ 3 seconds per generation with quantitative traits and selection. Decreasing the population size allows the simulation of a greater number of generations in a reasonable amount of time: for a population size of 10,000, it takes ~ 3 seconds per 100 generations (without selection, with a slight increase with selection). Configuration files for these benchmark scenarios are included in the `forqs` packages (`examples/benchmark_*.txt`).

To illustrate the simulation of entire genomes, the configuration file `examples/example_qtl_random_full_human_genome.txt` specifies a scenario where two populations (each size 10,000) are selected for different optimum values of a quantitative trait (Fig-

ure 3.9). In this simulation, individuals have human-sized genomes (23 chromosome pairs, chromosome length 100 million base pairs), with 50 QTLs placed randomly across the genome; the simulation takes ~ 2 seconds per generation (averaged over 50 generations).

Such a simulation is not feasible for forward-in-time simulators that store individual mutations in an array. For example, suppose that each locus for each individual chromosome is encoded in a single bit. Then to simulate 20,000 individuals as in the above scenario, 16GB of memory (available on high-end computers) will permit the simulation of only 3.2 million base pairs of sequence. This is an order of magnitude smaller than the smallest human chromosome (or roughly 0.1% of the human genome).

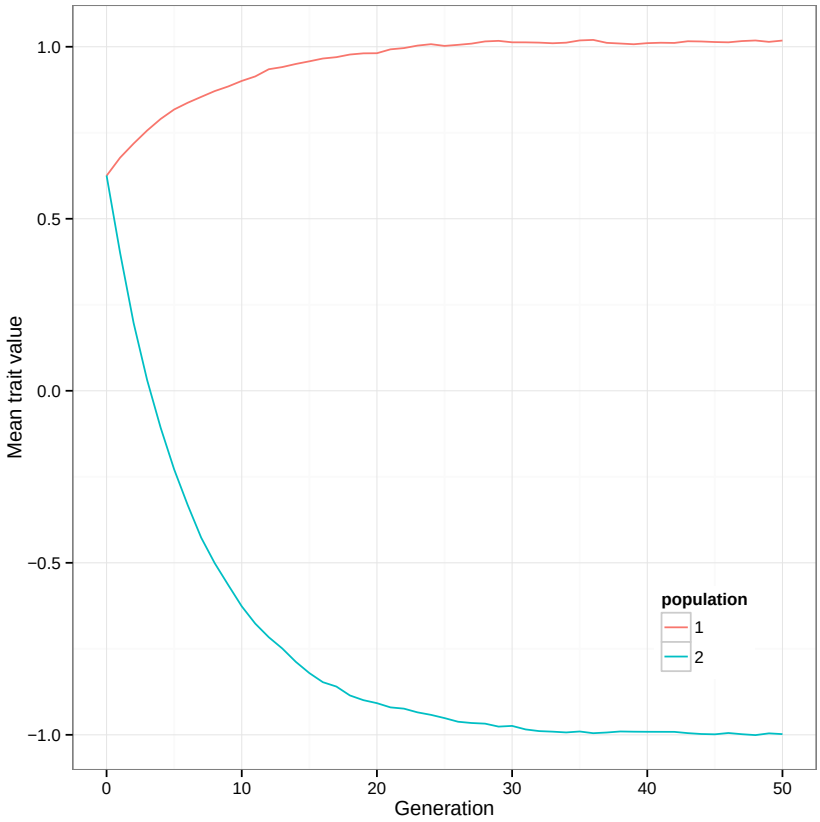

Figure 3.9: **Selection for different optimal values of a quantitative trait.** Shown are mean trait values for two populations (each size 10,000) that are selected for different optimal trait values. In this scenario, the quantitative trait has 50 QTLs distributed randomly over 23 chromosomes, with effect sizes drawn from an exponential distribution.

3.10 Software development

3.10.1 Software dependencies

`forqs` is written in C++, and makes extensive use of the Boost libraries, including the Boost build system:

`http://boost.org`

forqs also uses the muparser library for parsing mathematical expressions. This is used in QuantitativeTrait_Expression, which allows the user to specify quantitative traits that depend on other traits via a mathematical formula. The muparser source code is included in the forqs codebase, and is built automatically with forqs . The muparser library web-page is here:
http://muparser.beltoforion.de

3.10.2 Building the program

forqs is regularly built and tested on OSX and Linux. In addition, Windows binaries are built using cross-compilation with gcc on Linux.

The code can be obtained via git from bitbucket:

```
git clone https://bitbucket.org/dkessner/forqs.git
```

If you have the both the Boost libraries and the Boost build system installed, you can build forqs with the bjam tool, which is the Boost build system's equivalent of make:

```
cd forqs/src
bjam
```

3.10.3 Structure of the codebase

Most code units in the forqs project consist of three files:

- interface (header file)
- implementation (cpp file)
- unit test (cpp file)

For example, the code for the `Locus` module can be found in the files `Locus.hpp`, `Locus.cpp`, `LocusTest.cpp`.

The unit test for each code unit is run during the build; the build is successful only if all unit tests pass.

In addition to the unit tests, there are also regression tests, consisting of scripts in the `regression_test` directory. These tests prevent unintended behavior changes in the software during new development. Most of the tests are based on the example configuration files in the `examples` directory, and consist of running `forqs` and performing a `diff` comparison with known output. The regression tests are controlled by a Makefile, and run by `make`.

The codebase includes all documentation, including the Latex source code for this document. The `forqs` Module Reference is generated by Doxygen from the source code – each module is documented using Doxygen markup in the module's header file, immediately preceding the module's class declaration.

3.10.4 Software architecture

This section describes some of the design details of `forqs` that may be of interest to programmers who would like to add new features.

3.10.4.1 Low-level data structures

These are the basic low-level data structures used by `forqs`:

- `HaplotypeChunk`: a pair of integers (*position*, *id*)
- `Chromosome`: an array of `HaplotypeChunks`
- `Organism`: an array of pairs of `Chromosomes`
- `Population`: an array `Organisms`

In this setup, `Population` can be seen as an array of arrays, which requires a significant number of memory allocations. To avoid these memory allocations, `Population` was re-implemented as a two dimensional array of `Chromosome` pairs, resulting in a roughly 30% speedup.

The implementations produce identical results, and can be used interchangeably. To accomplish this, `Population` is actually an interface class, with two concrete implementations: `Population_Organisms` and `Population_ChromosomePairs`. While the latter is faster, the former can be more convenient for testing purposes.

In order to handle both cases in a unified manner, the class `ChromosomePairRange` represents the begin/end of chromosome pairs belonging to a single individual, and `ChromosomePairRangeIterator` does the appropriate iteration through individuals, depending on the memory layout of the population.

3.10.4.2 Configurable modules

Configurable modules in `forqs` are represented by the `Configurable` abstract base class, which defines the common interface through which objects are instantiated and initialized based on parameters specified by the user. The top-level module (`SimulatorConfig`), all primary modules (e.g. `PopulationConfigGenerator`, `Reporter`, etc.), and all building block modules (e.g. `Trajectory`, `Locus`) implement the `Configurable` interface.

The `Configurable` interface describes the functionality required to serialize an object's configuration, which must be representable as a list of name-value pairs. (This is not actually very restrictive, since the value can be anything representable as a string, including an array of numbers).

`Configurable` objects are instantiated by `SimulationBuilder_Generic`, which handles the mapping from the module name to the actual C++ class. Each object in the user-

specified configuration file is instantiated and registered by name so that other objects can obtain references to it if necessary. `SimulationBuilder_Generic` produces the final `SimulatorConfig` used by `forqs` for the simulation. (Note on the name: there were other `SimulationBuilder` classes that constructed `SimulatorConfigs` based on a more limited set of parameters – these have since been deprecated in favor of the more generic specification via configuration files.)

After instantiation, `Configurable` objects are configured/initialized in two steps. First, the `configure()` method sets any parameters specific to this module that are specified by the user. Second, after all objects have been configured, the `initialize()` method allows the modules to communicate with each other. For example, objects that need information about the lengths of the chromosomes (e.g. for recombination) can query the `PopulationConfigGenerator`, which has this information.

`Configurable` objects are instantiated, configured, and initialized in the order they are specified in the configuration file. It is important to note that while objects can obtain references to previously instantiated objects during the `configure()` step, they should not try to communicate via these references until the `initialize()` step, since the objects they refer to may not be fully initialized until then.

The main `Simulator` object interacts with `Configurable` objects through intermediate interfaces. This allows the main simulation logic to be separated from the specific behavior implemented by the various `Configurable` modules. For example, `Reporter_AlleleFrequencies` and `Reporter_TraitValues` both implement the `Reporter` interface. `Simulator` will give each `Reporter` information about the current populations, but doesn't need to know what they do with this information. As another example, the `Simulator` gives a `QuantitativeTrait` object information about individuals' genotypes and expects to receive trait values for each individual, but doesn't need to know details about how this is accomplished.

The `Configurable` interface also facilitates some features that make `forqs` more user-friendly, because it provides a translation layer between the user and program internals. For example, while 0-based indexing is used internally, from the user's perspective chromosomes and populations are numbered starting with 1. Also, `Configurable` modules can support multiple alternate parametrizations. For example, linear trajectories may be parametrized by slope-intercept or by two endpoints, and exponential distributions may be specified by the mean or the rate, depending on which is more natural for a particular scenario.

3.10.4.3 Mutation handling

Because `forqs` tracks haplotype chunks rather than sequences of variants, mutation is necessarily more complicated than recombination. This is because a new point mutation essentially creates a new haplotype that is identical to the original except at the mutated site. In order to accomplish this, the `VariantIndicator` must be able to be updated with the new haplotype and variant value. In addition, the haplotype's ancestry must be stored, since that haplotype chunk may contain other variants known by the `VariantIndicator`. This is implemented with the special `VariantIndicator_Mutable` that is not specified by the user. Instead, it is a wrapper class that is automatically instantiated when the user specifies a `MutationGenerator`. This wrapper class provides the functionality for updating variants, but passes calls through to the user-specified `VariantIndicator` for non-mutated loci.

3.10.5 Appendix: Boost libraries

The Boost C++ Libraries are an essential extension to the C++ Standard Library. In addition to providing functionality missing from the Standard Library, the Boost `filesystem` library and build system insulate the programmer from many platform-specific details.

If you have administrative access to your computer, you can use a package manager to install the Boost libraries and build system. If not, you can install Boost locally in your home directory.

The following are instructions for doing this on a Unix-like system (e.g. OSX or Linux):

```
# download latest Boost package, uncompress

# install boost libraries

cd boost_???                        # go into the uncompressed directory
bootstrap.sh --help                 # see options
bootstrap.sh --prefix=$HOME/local   # install in ~/local
./b2                                # builds libraries -- get coffee (this can take a while)
./b2 install                        # installs stuff in ~/local/include/boost and ~/local/lib

# install boost build

cd tools/build/v2
./bootstrap.sh
./b2 install --prefix=$HOME/local   # puts bjam in ~/local/bin, boost-build in ~/local/share

# also:
#   put ~/local/bin in your path (to find bjam)
# you might need this in your environment:
#   export BOOST_BUILD_PATH=$HOME/local/share/share/boost-build

# to avoid bjam warning: No toolsets are configured.
# create the file user-config.jam in either ~ (home) or ~/local/share/boost-build
# with the following line (note the space before ; is necessary):
    using gcc ;
```

3.10.6 Appendix: Cross-compilation targeting Windows

Cross-compilation of Windows binaries from Linux can be done using tools from the `mingw-w64` project (`http://mingw-w64.sourceforge.net/`). On Debian-based systems (e.g. Ubuntu, Mint), installing package `g++-mingw-w64` will install the necessary tools and dependencies.

```
# add this line to user-config.jam, which defines the toolset gcc-windows
# (you need to tell Boost build which g++, ar and ranlib to use):
```

```
using gcc : windows : i686-w64-mingw32-g++ : <archiver>i686-w64-mingw32-ar
    <ranlib>i686-w64-mingw32-ranlib ;

#   build the Boost libraries (or at least filesystem and system) for Windows
#   (you may need to fiddle with these command line parameters -- these worked for me),
#   from your Boost source dir:
bjam toolset=gcc-windows --prefix=$HOME/local_win32 threading=multi
  target-os=windows link=static threadapi=win32 --without-mpi
  runtime-link=static --without-python -sNO_BZIP2=1 --layout=tagged

# note: LIBRARY_PATH doesn't appear to work for cross-compilation, but you can
# put the Boost library files directly in /usr/i686-w64-mingw32, where the various
# development headers/libs are installed for mingw-w64

# the forqs Jamroot contains a target ``windows'', which will build Windows binaries
# and put them in forqs/bin_windows:
bjam windows
```

CHAPTER 4

Power analysis of artificial selection experiments using efficient whole genome simulation of quantitative traits

4.1 Abstract

Evolve and resequence studies combine artificial selection experiments with massively parallel sequencing technology to study the genetic basis for complex traits. In these experiments, individuals are selected for extreme values of a trait, causing quantitative trait loci (QTLs) to increase or decrease in frequency in the experimental population. We present a new analysis of the power of artificial selection experiments to detect and localize quantitative trait loci. This analysis uses a novel simulation framework that explicitly models whole genomes of individuals, quantitative traits, and selection based on individual trait values. We show that modeling loci with constant selection coefficients does not fully capture the dynamics of QTLs under artificial selection. We also show that a substantial portion of the genetic variance of the trait (50–100%) can be explained by detected QTLs in as little as 20 generations of selection, depending on the trait architecture and experimental design. Furthermore, we show that power depends crucially on the opportunity for recombination during the experiment. Finally, we show that an increase in power is obtained by leveraging founder haplotype information to obtain allele frequency estimates.

4.2 Introduction

Some of the first artificial selection experiments, performed in the early 1900s, were designed to settle ongoing debates about the nature of selection. In particular, early researchers hoped to answer questions about whether selection on continuous variation was even possible, and how to reconcile this with the Mendelian viewpoint of genes as discrete heritable units (see Falconer (1992) for a history of early selection experiments, and Garland and Rose (2009) for a comprehensive overview of experimental evolution).

More recently, researchers have used artificial selection experiments to study the genetic basis for complex traits by analyzing allele frequency changes in evolved populations. This technique has been used in a wide variety of organisms, including for example maize (Laurie *et al.*, 2004), yeast (Ehrenreich *et al.*, 2010), fruit flies (Nuzhdin *et al.*, 2007; Teotonio *et al.*, 2009), chickens (Johansson *et al.*, 2010), and mice (Keightley and Bulfield, 1993).

Massively parallel (also known as next-generation) sequencing of pooled samples has enabled researchers to obtain genome-wide allele frequency estimates for a population in a cost-effective manner (Futschik and Schlotterer, 2010). These technological advances have led to the development of the Evolve and Resequence (E&R) method for mapping traits (Burke *et al.*, 2010; Turner *et al.*, 2011; Parts *et al.*, 2011; Orozco-terWengel *et al.*, 2012; Remolina *et al.*, 2012). In E&R studies, artificial selection is followed by genome-wide pooled sequencing. In the case of a quantitative trait, a site exhibiting a large allele frequency change in the selected population suggests the presence of a nearby quantitative trait locus (QTL) (Figure 4.1).

Due to limitations on resources and time, researchers will face many design decisions as they set up artificial selection experiments. First, the size of experimental populations affects the degree of genetic drift and the efficacy of selection. Next, the strength of selection is determined by the proportion of individuals selected each generation to create the next generation. In addition, the length of the experiment must be chosen in conjunction with the strength of selec-

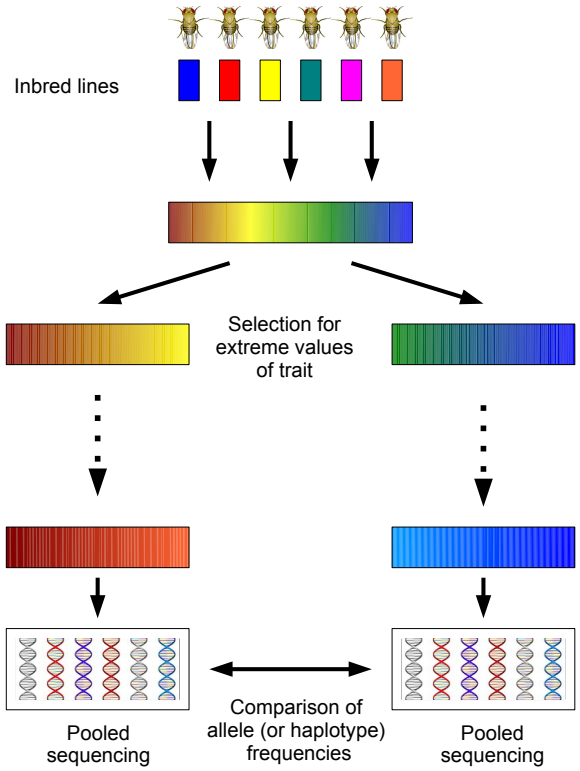

Figure 4.1: **Evolve and resequence experiment.** After initial neutral mixing of founders, individuals with extreme values of the trait are selected to create the next generation. After several generations of selection, populations are sequenced and analyzed for allele frequency differences.

tion so as to maximize allele frequency differences at QTLs, while at the same time minimizing differences at neutral loci due to drift. One concern is that extremely strong selection over a large number of generations will result in the fixation of variants across the genome, which will stifle any ability to distinguish QTLs from neutral loci. In the case of selection on a quantitative trait, selection can be performed on a single population which will be compared to a control population; alternatively, selection can be divergent, where two populations selected in opposite directions will be compared. Finally, replication is a key consideration in any experiment, and

especially in experimental evolution, where random genetic drift plays a large role: observation of a large allele frequency difference at a locus in multiple replicate experiments will increase confidence that the difference is due to selection at the locus rather than drift.

Previous studies of the power of artificial selection experiments to detect trait loci have taken the traditional population genetics approach in which selection is parametrized using selection coefficients that remain constant each generation. For example, previous work by Kim and Stephan (1999) analyzed a single locus under a constant selection coefficient using a diffusion approximation, and two recent simulation studies employed forward simulations with loci parametrized by constant selection coefficients to obtain power estimates (Kofler and Schlotterer, 2014; Baldwin-Brown *et al.*, 2014).

However, E&R experiments lie at the interface between population genetics and quantitative genetics. The selection pressure on a QTL is more naturally parametrized using concepts from quantitative genetics such as effect sizes, genetic variance, and heritability, in addition to experimental design parameters such as the proportion of individuals selected each generation. Furthermore, as we will see, the effect of a QTL allele on an individual's fitness depends strongly on interaction with the other QTL alleles carried by the individual. As noted by Felsenstein (1974, 1987), finite populations necessarily experience random linkage disequilibrium between any two polymorphic loci. This linkage disequilibrium results in interference, where the effect of selection on each locus is decreased (Hill and Robertson, 1966).

In this study, we introduce a new simulation framework to investigate the power of artificial selection experiments to detect and localize QTLs contributing to a quantitative trait. Our simulations employ a whole-genome quantitative genetic model of loci underlying a quantitative trait, and we explicitly model artificial selection of individuals each generation based on their trait values. As in Kofler and Schlotterer (2014) and Baldwin-Brown *et al.* (2014), we model *Drosophila melanogaster* individuals, and we use experimental parameters similar to those used by Turner and Miller (2012).

Our simulations show that forward simulations of a locus assuming a constant selection coefficient do not fully capture the allele frequency dynamics of a QTL under artificial selection on a quantitative trait. In contrast, explicit quantitative genetic modeling of the trait leads to insights regarding the effect of the trait architecture on the allele frequency trajectory of a QTL. For instance, by simulating the entire genome of individuals, we also demonstrate the important role that recombination plays in decreasing interference between QTLs, as well as reducing linkage disequilibrium between QTLs and neighboring neutral loci. We show that designing the artificial selection experiment to allow more opportunity for recombination increases the ability to detect and localize QTLs.

Finally, previous work suggested that when founder sequence information is available, one can obtain more accurate allele frequency estimates by estimating local haplotype frequencies from pooled read data (Kessner *et al.*, 2013). We show that these improved allele frequency estimates, when compared to estimates calculated directly from read counts, lead to an increase in power.

4.3 Results

4.3.1 Comparison between selection coefficient simulations and explicit quantitative genetic modeling

To address the importance of modeling quantitative traits, we first investigated allele frequency trajectories of a focal QTL contributing to a quantitative trait under truncation selection in comparison to a single locus with a constant selection coefficient.

We simulated populations of size $N = 1000$ for the selection coefficient simulations. For the artificial selection simulations, we simulated populations of size $N = 5000$ with 20% selected to create the next generation, so that the effective population size was $N_e = 1000$. We ran the artificial selection simulations with 12 canonical trait architectures with various heritabilities and

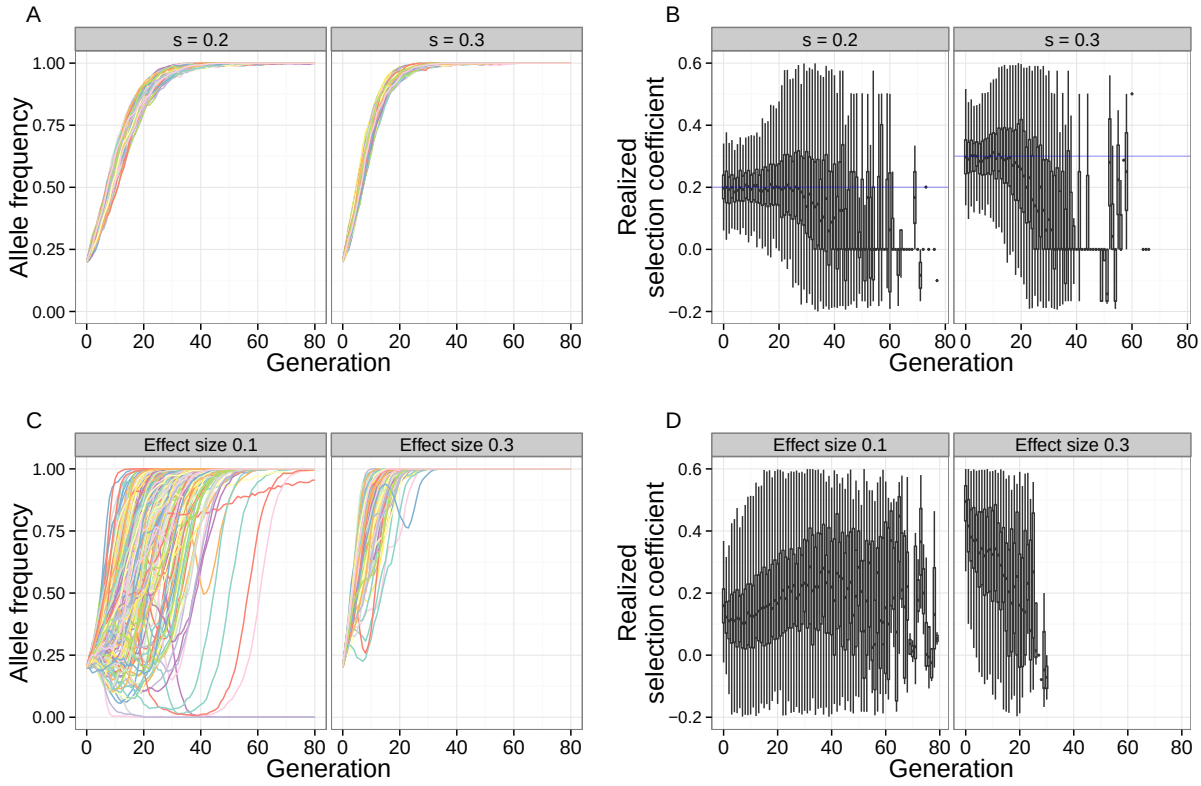

Figure 4.2: **Qualitative differences between fixed selection coefficient and truncation selection on a quantitative trait.** A focal QTL exhibits fundamentally different behavior under a constant selection coefficient, compared to truncation selection on a quantitative trait. Shown are allele frequency trajectories (A) and realized selection coefficient distributions (B) of a locus under two different selection coefficients (.2, .3) [N=1000, 80 generations]. C and D show the same for a focal QTL (effect sizes .1, .3) under truncation selection on the trait [N=5000, 20% selected, 80 generations, 100 QTLs, $h^2 = .8$, $\sigma^2_{trait} = 1$].

QTL counts, and with exponentially distributed effect sizes (see Methods). In all simulations, the initial frequency of trait-increasing allele was .2, and selection is assumed to be in the direction of increasing trait values.

As seen in Figure 4.2 (A and C), the allele frequency trajectories of the focal QTL under

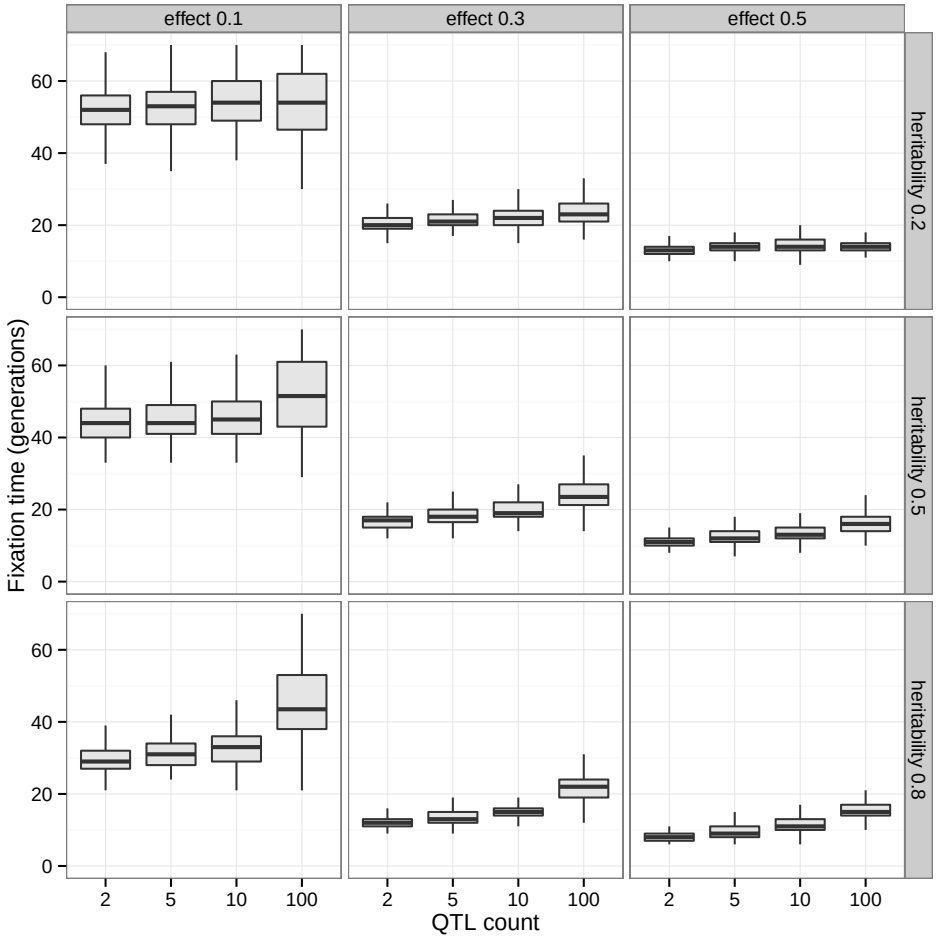

Figure 4.3: **Effect of genetic architecture on fixation times** Fixation times of a focal QTL for a trait under truncation selection decrease with increasing effect size and heritability. For a fixed effect size and heritability, the fixation times increase with the number of QTLs contributing to the trait due to interference. [N=5000, 20% selected, 80 generations]

truncation selection are qualitatively different from the trajectories under a constant selection coefficient. While a strong selection coefficient leads to nearly deterministic allele frequency trajectories, trajectories of a focal QTL under strong truncation selection are dependent on the underlying trait architecture. In particular, the focal QTL may experience interference from

other linked QTLs. This effect can be seen in the trajectories where the focal QTL decreases in frequency at first, due to repulsion linkage (with a QTL of opposite effect), but increases in frequency once linkage disequilibrium has decreased sufficiently through recombination.

In addition, once an allele with a constant selection coefficient reaches high frequency, it remains at high frequency for several generations before finally going to fixation. In contrast, the focal QTL under truncation selection tended to become fixed quickly after reaching high frequency in the population. This behavior is not surprising, because after a few generations of selection, the upper tail of the population trait value distribution is highly enriched for individuals carrying high-effect variants.

To further illustrate these qualitative differences, we analyzed the realized selection coefficient of the trajectories, which represents the selection coefficient that would result in a given single-generation allele frequency change under a deterministic model (see Methods). Under a constant selection coefficient, the mean realized selection coefficient tracks the true selection coefficient closely during the selection phase, after which it decreases to zero during the drift phase (Figure 4.2B). Under truncation selection, the behavior of the mean realized selection coefficient depends on the underlying genetic architecture of the trait. When the effect size is low, the realized selection coefficient increases each generation – this is due to selection acting on the larger effect QTLs first, and then having a greater effect on the focal QTL after the larger effect QTLs have reached fixation. On the other hand, when the effect size is higher, the focal QTL experiences very strong selection initially, decreasing as the focal QTL rises to fixation (Figure 4.2D).

Another effect of the underlying trait architecture can be seen in the fixation times of the focal QTL (Figure 4.3). Because truncation selection will have a stronger effect on QTLs with a large effect size, the fixation time of the focal QTL decreases as the effect size increases. On the other hand, for a given effect size and heritability, the fixation time of the focal QTL increases with total number of QTLs due to interference.

4.3.2 Measurement of power to detect and localize QTLs

In all of the following analyses, we examine the power to detect QTLs through allele frequency differences at variant sites. There is currently no consensus regarding the choice of statistic in the analysis of artificial selection experiments. Both for simplicity and becuase of its use in practice (Parts *et al.*, 2011; Turner and Miller, 2012) we chose to base our analysis on the absolute allele frequency difference $D = |p_1 - p_2|$, where p_1 and p_2 are the allele frequencies of the variant in populations 1 and 2, respectively. We also note that this statistic, also called the difference in derived allele frequencies (ΔDAF or $DDAF$), is used in several related or composite methods for detecting selection (Turner *et al.*, 2011; Grossman *et al.*, 2010; Utsunomiya *et al.*, 2013).

We calculate D for each variant site in the genome, and we call a site detected if the D value exceeds a threshold value. By varying the threshold, we obtain receiver operating characteristic (ROC) curves showing the relationship between power (true positive rate) and the false positive rate.

Due to linkage and strong selection, detected QTLs will generally have neighboring neutral variants whose allele frequency differences also exceed the detection threshold. Because of this, detection and localization of a QTL are necessarily intertwined. In an actual experimental setting, the entire genomic region surrounding the significantly diverged loci would be chosen for follow-up studies.

We explored several methods for calculating and interpreting power and false positive rate. We present our results using the method that we found to be most interpretable, and which we summarize here (see Methods for full details on the different methods). For a given D value threshold, we determine a *detection region* that consists of all variants within a specified radius of any variant above the threshold (10kb for the results presented here; see Figure 4.9 for an illustration). Power is calculated as the proportion of genetic variance (in the founder population) expained by the QTLs located within the detection region. We define the *true region* to consist of all variants within the specified radius of any QTL, and we define the *neutral genome* to be

the rest of the genome (outside the true region). False positive rate is calculated as the proportion of the neutral genome covered by the detection region. Thus, a false positive rate of .01 can be interpreted to mean that 1% of the genome would be incorrectly flagged for follow-up studies. In *Drosophila melanogaster*, this would correspond to a 1.4mb region.

4.3.3 Advantages of divergent artificial selection

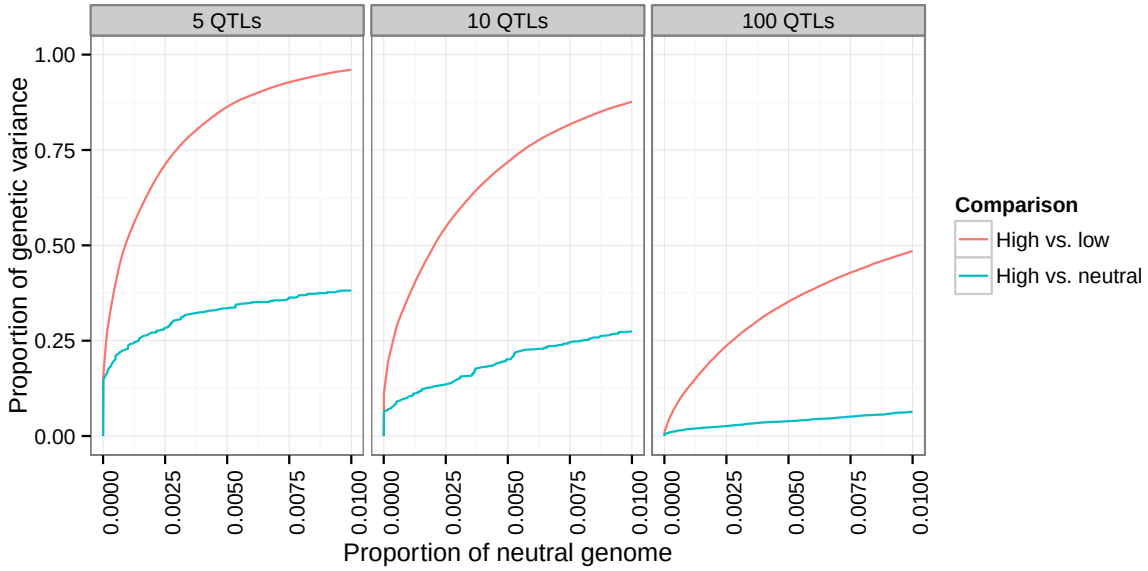

Figure 4.4: **Increase in power due to divergent selection.** Comparison of two populations divergently selected for extreme values of a trait has greater power to detect QTLs than comparison between selected and control populations. Three populations of 1000 individuals (high, low, neutral) originating from a single founder population were simulated for 20 generations, with 20% selected each generation in the high and low populations. Shown are the scenarios with 5, 10, and 100 QTLs, with $h^2 = .5$.

In artificial selection studies of quantitative traits, one commonly used technique is divergent (also called bidirectional) selection, where the first "high" population is obtained by selecting

from the upper tail of the trait value distribution each generation, and the second "low" population is obtained by selecting from the lower tail (for example Johansson *et al.* (2010) and Turner and Miller (2012)). Use of this technique presumes that allele frequency differences between the high and low lines will be more pronounced at QTLs contributing to the trait than, for example, differences between the high line and a control population that has been evolving neutrally.

To compare the power obtained by a divergent selection experiment to the power obtained by selection in a single direction, we simulated 3 populations originating from a single founder population, where one population was selected for high values, one selected for low values, and one allowed to evolve neutrally. Each population had 1000 individuals, of which 20% were selected each generation in the high and low populations, for 20 generations total.

We ran the simulations for each of our 12 canonical architectures (see Methods), and we show the results in Figure 4.4 for some of these ($h^2 = .5$, with 5, 10, and 100 QTLs). Comparison between the high and low populations leads to a substantial increase in power over the comparison between the high and neutral populations. In particular, at a 1% false positive rate, roughly half of the genetic variance of the trait is detected in the high vs. low comparison but not in the high vs. neutral comparison.

4.3.4 Extent of increased power due to replication

Another technique available in artificial selection experiments is the use of replicate high and low populations to increase confidence that allele frequency differences between diverged populations are due to selection rather than genetic drift. A natural generalization of the allele frequency difference D for replicate populations is $D = |\sum_{i=1}^{R} p_{i,high} - p_{i,low}|$, where R is the number of replicate population pairs.

We investigated the effects of evolving replicate populations by simulating 5 pairs of populations originating from a single founder population [10 QTLs, $h^2 = .5$, N=1000, 20% selected, 20 generations]. We then calculated the average power to detect QTLs using subsets of the data

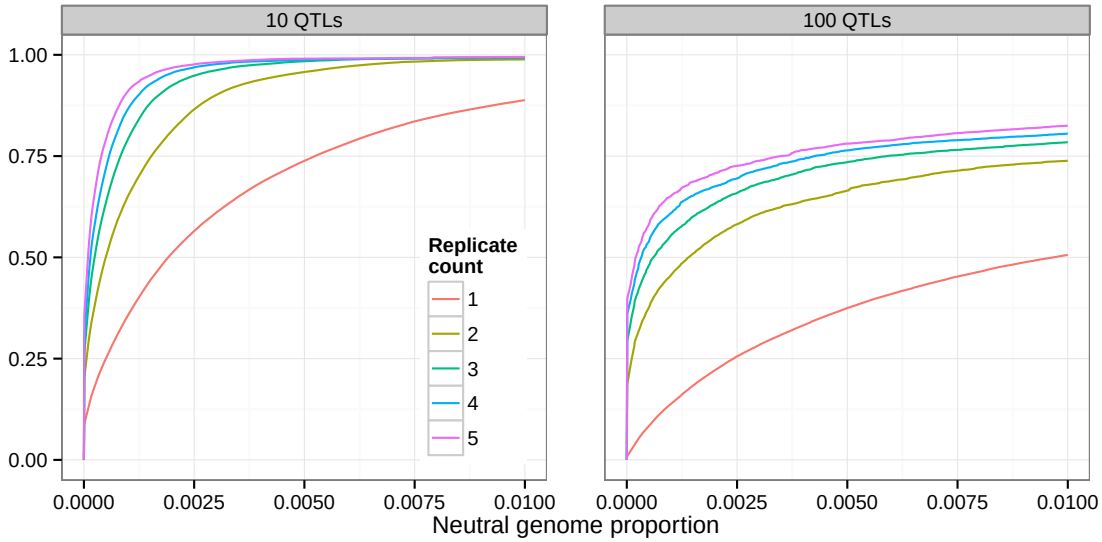

Figure 4.5: **Increase in power due to replicate populations.** Adding replicate pairs of divergently selected populations increases power to detect QTLs. 5 pairs of populations originating from a single founder population were simulated [$h^2 = .5$, N=1000, 20% selected, 20 generations].

representing population replicates from 1 to 5 pairs.

We found that using 2 replicate populations substantially increases power to detect QTLs (Figure 4.5). For example, at the low false positive rate (neutral genome proportion) of .0005, the proportion of genetic variance detected increases from 25% to 50%. Adding further replicate populations continues to increase power, but with diminishing returns.

4.3.5 Effect of population size

It is well known that selection on a single locus, defined by a selection coefficient s, acts more efficiently in larger populations, as can be seen in the dependence of fixation probabilities and fixation times on the population-scaled selection coefficient $4Ns$ (Ewens, 2004).

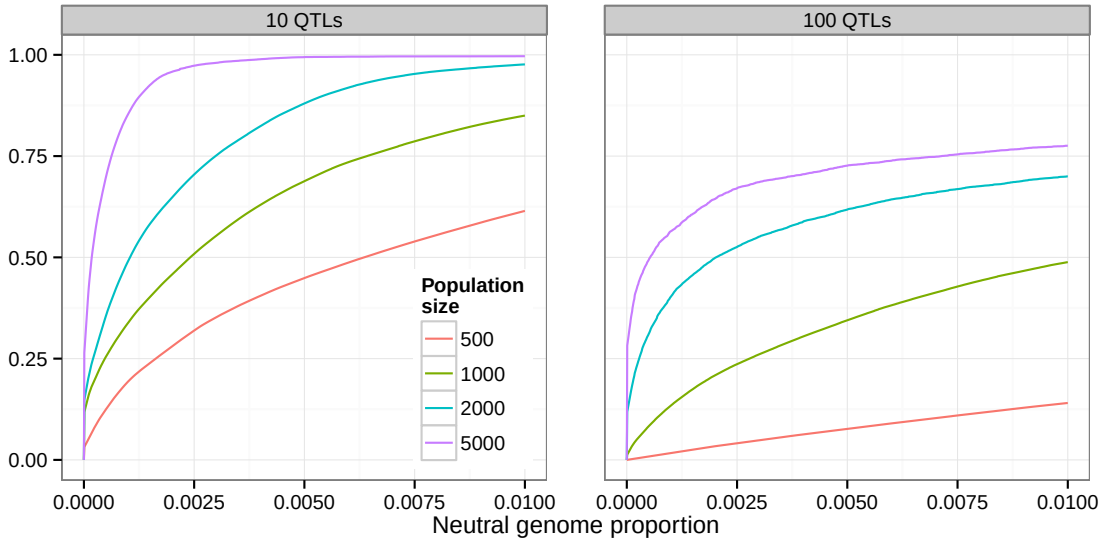

Figure 4.6: **Increase in power due to population size.** Using larger population sizes substantially increases power to detect QTLs [$h^2 = .5$, 20% selected, 20 generations].

To investigate how the population size affects artificial selection experiments, we performed simulations of populations of different sizes ($N = 500, 1000, 2000, 5000$) under identical experimental conditions (20% selected, 20 generations), with each of our 12 canonical trait architectures.

In all cases we found a substantial increase in power to detect and localize QTLs as we increased the population size. For example, in the case of 10 QTLs and $h^2 = .5$, increasing the population size from 500 to 1000 resulted in an additional 25% of the genetic variance detected at the $.5\%$ false positive rate (Figure 4.6). Moreover, while less than 50% of the genetic variance was detected with $N = 500$ at that false positive rate, nearly 100% was detected with $N = 5000$.

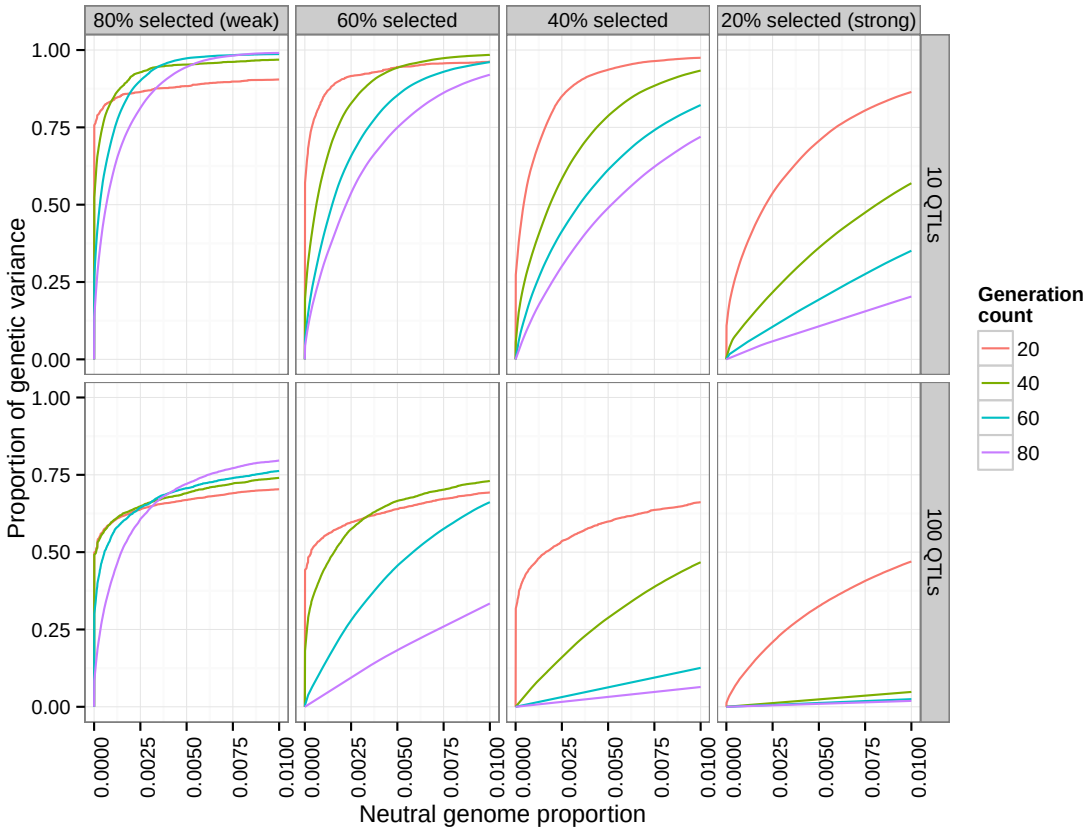

Figure 4.7: **Effects of length of experiment and selection strength.** Experiments with weak selection (top/bottom 80% selected each generation) over a longer number of generations have higher power than those with strong selection (20% selected). [N=1000, $h^2 = .5$]

4.3.6 Effects of the length of experiment and proportion selected

We next investigated the effects of the length of the experiment and the strength of selection on the power to detect QTLs. To do this, we simulated 4 different selection scenarios, from weak selection (top/bottom 80% selected each generation) to strong selection (20% selected). Each simulation ran for 80 generations, and we examined snapshots of each population at generations 20, 40, 60, and 80. In these simulations, the populations consisted of 1000 individuals, and the trait had 10 QTLs, with a heritability of .5.

One particularly interesting finding was that under strong selection, power can actually decrease with the number of generations, depending on the false-positive rate threshold chosen (for example, 20% or 40% selected, at false-positive rate $.01$ in Figure 4.7). In these scenarios, most of the QTLs have differentiated by generation 20 (over 75% of the genetic variance was detected at false positive rate 1%). However, the lower effective population size induced by strong selection results in the fixation of many neutral variants, which are then falsely detected as QTLs.

Conversely, by lowering the selection pressure (80% selected each generation) the maximum power is actually increased. In addition, letting the experiment run for a greater number of generations at this lower selection pressure increases the maximum power attained.

These observations suggest that recombination plays a large role in the power to detect and localize QTLs: reducing the selection pressure and increasing the number of generations should allow more recombination, which reduces linkage between QTLs and neighboring neutral variants. Increased recombination will also reduce interference between QTLs, which should allow lower effect QTLs to be selected and detected. This led us to investigate the effects of recombination further in our subsequent analyses.

4.3.7 Effect of recombination

To investigate the effects of recombination on the power to detect and localize QTLs, we first considered the effect of increasing the recombination rate of the simulated individuals. We simulated populations of 1000 individuals, with each our 12 canonical trait architectures, with recombination rates at 1x, 2x, 3x, and 4x our standard recombination rate (see Methods) [20 generations of neutral mixing, 20 generations of selection, 20% selected].

We found that increasing the recombination rate does indeed increase the power to distinguish QTLs from neutral variants, for all trait architectures (Figure 4.8AD). While in practice it is not feasible to experimentally increase the recombination rate of individuals, the experiment

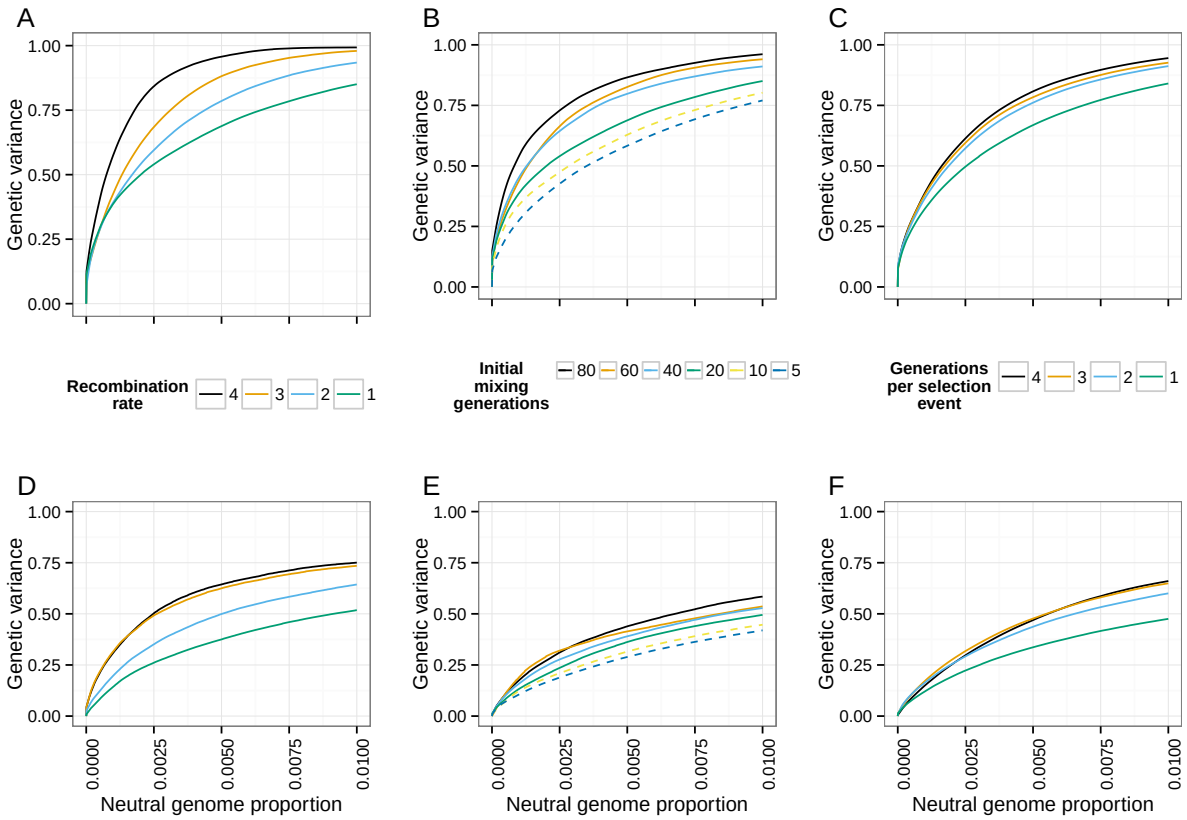

Figure 4.8: **Effect of recombination.** The power to detect and localize QTLs depends on the extent of recombination experienced by the populations. (A,F) Power increases with the recombination rate of the organisms under selection. (B,E) Additional generations of initial mixing and (C,F) additional generations between rounds of selection similarly increase power. [top row (A-C): 10 QTLs, bottom row (D-F): 100 QTLs, $h^2 = .5$, N=1000, 20% selected, 20 generations of selection, 20 generations of mixing in A, C, D, F]

can be designed so as to increase the opportunity for individual chromosomes to recombine and thus decrease linkage between QTLs and neighboring neutral variants.

One way to decrease linkage disequilibrium in the population is to allow more generations of initial neutral mixing in the founder population, before selection starts (e.g. Turner and Miller

(2012)). We simulated scenarios with varying numbers of generations of neutral mixing (20, 40, 60, 80), with the other parameters as above. We found that increasing the number of generations of initial neutral mixing increases power (Figure 4.8BE).

An alternative way to decrease linkage disequlibrium is to intersperse extra generations of neutral mixing between rounds of selection. To test this idea, we simulated scenarios where selection was carried out every 1, 2, 3, or 4 generations, with random mating during the generations where selection did not take place (other parameters as above). Again, we found that the additional recombination events increased power to detect QTLs (Figure 4.8CF). We further illustrate the effect of extra recombination on allele frequency differences in Figure 4.9.

Taken together, these results show that the ability to detect and localize QTLs depends crucially on recombination, both to decrease interference between QTLs and to break up linkage between QTLs and nearby neutral variants.

4.3.8 Haplotype-based inference of allele frequencies increases power

In all of our power analyses, we have assumed that we are able to calculate the allele frequencies at all variant sites perfectly. However, in actual experiments, the allele frequency at a locus will be estimated by considering the sequence read counts that cover the site. These estimates are prone to errors from two sources: random base errors, and stochasticity in the number of reads covering the site. The error in measurement of allele frequencies will lead to a loss of power to detect and localize QTLs.

In the case where founder sequences are known, our previous work (Kessner *et al.*, 2013) showed that one can leverage haplotype information from the founders to obtain local haplotype frequencies, and that these estimates are robust to both sequence read errors and uneven coverage. This work suggested that improved allele frequency estimates could be derived from local haplotype frequency estimates.

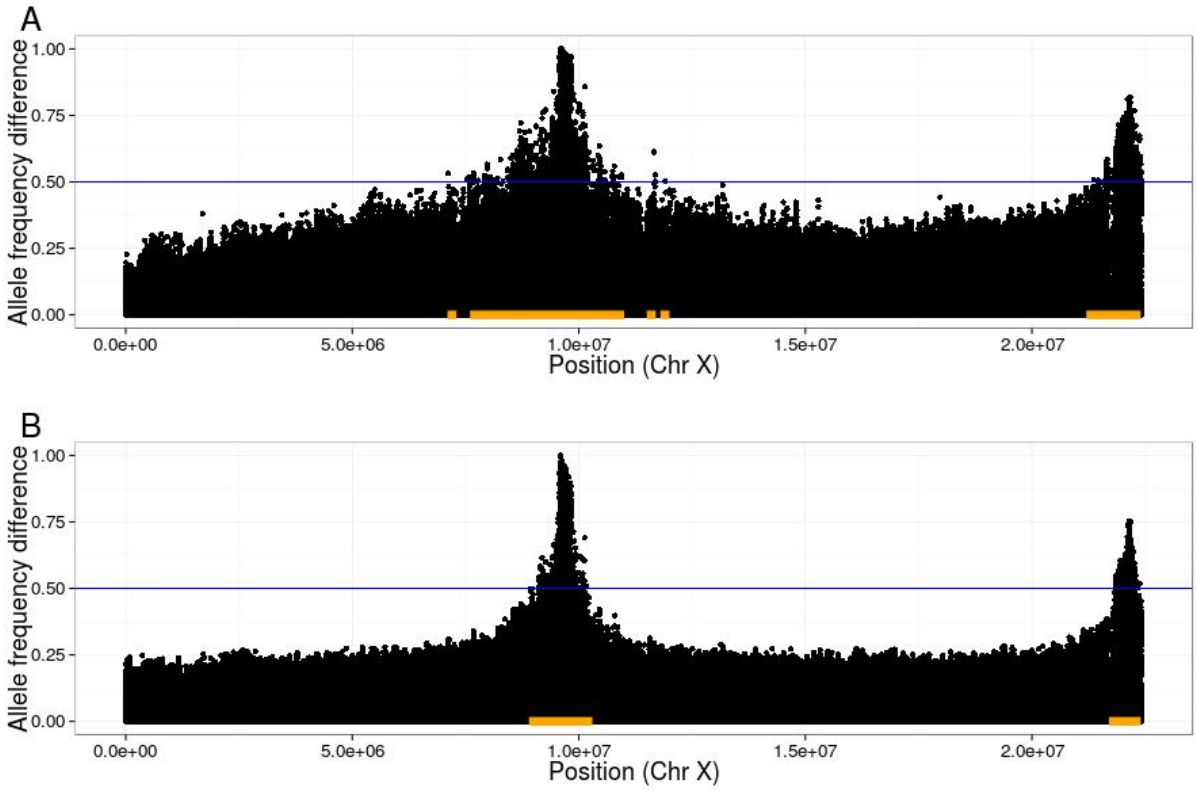

Figure 4.9: **Illustration of linkage disequilibrium between QTLs and neutral loci.** In plots of allele frequency differences between high and low populations, QTL peaks are narrower when extra generations of neutral mixing are introduced. As an example, we indicate a threshold of .5 with the blue lines, and the corresponding detection regions with orange bars. Note that the true region, consisting of sites within 10kb of either SNP, is too small to be seen at this scale; thus, the size of the orange bars represent the (local) false positive rate at this threshold. [10 QTLs, $h^2 = .5$, N=1000, 20% selected, 20 generations of initial mixing, shown are average D values over 20 replicates. A) 20 generations of selection B) 20 generations of selection, 4 generations per selection event (80 generations total)]

We investigated the effect of allele frequency estimation error on power by first obtaining empirical error distributions for the two estimation methods (see Methods). We then simulated

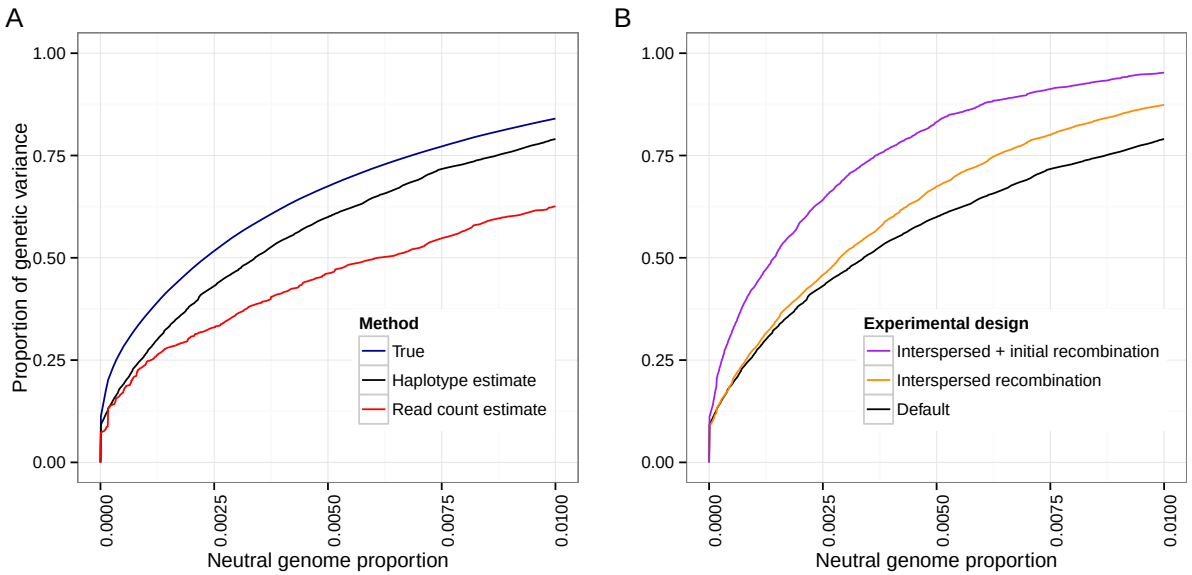

Figure 4.10: **Using founder haplotype information estimation improves power.** A) If founder sequence information is available, estimating local haplotype frequencies leads to better allele frequency estimates and an increase in power over estimates obtained from raw read counts. B) Error in local haplotype frequency estimates increases with the number of generations due to recombination; however, there is a still a net gain in power from extra generations of neutral mixing. Default: 20 generations mixing, 20 generations selection. Interspersed: 20 generations mixing, 4 generations per selection event, 20 selection events (100 generations total). Initial + interspersed: as interspersed, with 70 generations mixing (150 generations total). [10 QTLs, $h^2 = .5$]

artificial selection experiments with our standard experimental parameters [$N = 1000$, 20% selected, 20 generations] and canonical trait architectures, followed by the random introduction of errors based on the empirical error distributions. In all cases, we found that the haplotype-based allele frequency estimates led to improved power over the read-count-based estimates. Figure 4.10A shows the results from the case with 10 QTLs and $h^2 = .5$.

Our analyses from the previous section showed that increasing the amount of recombination led to an increase in power. On the other hand, increased recombination results in shorter haplotypes. This leads to greater errors in local haplotype frequency estimates and the allele frequency estimates derived from them, with a subsequent decrease in power. To investigate the relative magnitude of these two counteracting effects, we added two scenarios: one where we interspersed rounds of neutral mating between generations of selection (4 generations per selection event, 100 generations total for the experiment), and one where we also added an extra 50 generations of initial neutral mixing (150 generations total). We introduced random errors as above, using empirical error distributions that take into account the increased number of generations.

We found that recombination's positive effect of breaking up linkage between QTLs and neutral sites outweighed the negative effect on haplotype frequency estimation (Figure 4.10B). However, for experiments lasting 200 generations or longer, our haplotype-derived allele frequency estimates were no better than read-count-derived estimates (see Methods for parameter assumptions). Thus, we do not expect haplotype-derived allele frequency estimates to lead to increased power in scenarios where the typical length of haplotype chunks is shorter than the window size used for the local haplotype frequency estimation.

4.4 Discussion

We have presented a new analysis of the power of artificial selection experiments to detect and localize loci contributing to a quantitative trait. In this analysis, we explicitly model whole genomes of individuals, quantitative traits, and selection based on individual trait values, using a novel simulation framework.

We showed that population genetics simulations based on loci with constant selection coefficients do not fully capture the dynamics of QTLs contributing to a trait under artificial selection,

and that the trait architecture plays a large role in these dynamics. In addition, explicit modeling of selection on a quantitative trait has several other advantages. For example, simulated experiments can be parameterized and results can be reported using the standard quantitative genetics concepts of effect size, genetic variance, and heritability. Also, scenarios such as divergent selection can be simulated in a straightforward manner. Finally, the behavior of a QTL under artificial selection is dependent both on the experimental design (proportion of individuals selected each generation) and on the trait architecture (effect size, and linkage to other QTLs). While the selection coefficient conflates these parameters into a single number, our simulation framework allows these parameters to be separated and investigated independently.

Our results show the important role that recombination plays in the ability to identify QTLs. Recombination not only reduces interference between QTLs, but also decreases linkage disequilibrium between QTLs and neighboring neutral loci. In fact, one can view the artificial selection experiment as a classification problem. From this viewpoint, the ROC curve, which measures the ability to classify a locus as QTL vs. neutral, contains information about how well the experiment can break linkage disequilibrium between causal and neutral loci.

The classification viewpoint is useful when thinking about how to design a selection experiment. For example, we found that experiments allowing more opportunity for recombination, either during initial neutral mixing of the founder haplotypes or between selection events, have greater power to detect and localize QTLs. Similarly, experiments with weaker selection (greater proportion selected each generation) over a longer period of time will have greater power than shorter experiments with strong selection. We note that the improved mapping resolution afforded by additional recombination is analagous to the use of recombinant inbred lines (see Crow (2007) for a history) or advanced intercross lines (Darvasi and Soller, 1995) in traditional QTL mapping studies.

While recombination increases the ideal power of an artificial selection experiment, it also decreases the ability to use founder haplotype information to obtain more accurate allele fre-

quency estimates. We showed that the haplotype-based estimates still result in a net increase in power in experimental scenarios where the scale of recombination is larger than the window used for estimating local haplotype frequencies. This suggests that when choosing the window size for such an analysis, one should take into consideration the expected scale of recombination based on the experimental design.

Similar to the findings of Kofler and Schlotterer (2014) and Baldwin-Brown *et al.* (2014), we found that increasing population sizes and number of replicates leads to an increase in power. Additionally, our simulation framework allowed us to quantify the increase in power due to bidirectional selection. We also note that adding generations of initial neutral mixing in the founder population is in some ways similar to increasing the number of founder haplotypes, in that it places QTLs on multiple genetic backgrounds. Our results regarding the increase in power due to additional initial mixing are thus consistent with the findings of both of these groups that increasing the number of founder haplotypes increases power to detect and localize QTLs.

Also in agreement with Baldwin-Brown *et al.* (2014), but in contrast to Kofler and Schlotterer (2014), we found that the effect of increasing the length of the experiment is not uniformly beneficial, but rather depends on the strength of selection. From the classification viewpoint again, power depends on the ability of the experiment to allow QTLs to differentiate in selected populations while keeping allele frequencies at neutral loci constant. Continuing the experiment beyond the point where the majority of QTLs have differentiated will lead to increased fixation of linked neutral loci, and hence lower power.

In contrast to both Kofler and Schlotterer (2014) and Baldwin-Brown *et al.* (2014), whose results suggest that hundreds of generations are necessary to obtain reasonable power, we found that artificial selection experiments can detect QTLs explaining most of the genetic variance of the trait in as little as 20 generations, under reasonable assumptions about the trait architecture (100 loci, $h^2 = .5$, exponentially distributed effect sizes). We believe that the previous studies were perhaps conservative in their choice of ranges of selection coefficients. On the other hand,

it is not obvious how to interpret selection coefficients in the context of an artificial selection on a quantitative trait, where we feel it is more natural to use parameters representing the trait architecture and experimental design.

In summary, we have shown that it is feasible to do whole-genome simulations of artificial selection with explicit quantitative trait modeling. The scope of this study was limited to the use of *D. melanogaster* as our model species and simple trait architectures with no dominance or pleiotropic effects. However, we believe that our simulation methodology can be applied to a wide variety of species, trait architectures, and experimental protocols. We emphasize that the opportunity for recombination is a key factor in the power to detect and localize QTLs, and that this should be taken into account by future designers of artificial selection experiments.

4.5 Methods

4.5.1 Forward simulation

We used the program `forqs` (Kessner and Novembre, 2014) for all forward simulations. `forqs` simulates whole genomes of individuals efficiently by tracking the haplotype chunks that are inherited from the founder individuals in the initial generation. `forqs` allows the specification of quantitative traits, and provides flexible fitness functions for the simulation of complex evolutionary scenarios such as artificial selection.

In our simulations of artificial selection on a quantitative trait, individuals had 3 chromosomes, with lengths matching *Drosophila* chromosomes X, 2, and 3. For simplicity, we set the recombination rate to be uniform along each chromosome, with rate = 2 cM/mb, which is similar to recombination rates reported previously (Comeron *et al.*, 2012). For each set of experimental parameters we simulated populations using 200 replicates each of 12 different canonical architectures: 2, 5, 10, and 100 QTLs, at initial heritability levels .2, .5, and .8. For a given number of QTLs and heritability level, QTL positions and effect sizes were generated randomly for each

simulation run (see next section). The random trait generation was implemented in an auxilliary program which produces trait description files which are included by `forqs` configuration files that specify the experimental setup. Our simulation scenarios represented selection on standing variation where the expected waiting time $\frac{1}{2N\mu}$ for a mutation to occur at a QTL is much larger than the number of generations in the experiment. Hence, we did not include *de novo* mutations in our simulations.

Each generation, `forqs` calculates trait values for all individuals based on the effect sizes of the alleles they carry at QTLs and a random environmental effect (with variance determined by the heritability of the trait). Artificial selection was simulated using the `forqs` module `FitnessFunction_TruncationSelection`, which assigns a fitness value of 1 to those individuals whose trait value lies above (or below) a threshold (and 0 otherwise), where the threshold is determined each generation by the distribution of trait values in the population and the user-specified proportion of individuals selected to create the next generation.

All configuration files and analysis scripts for all simulations are freely available online: `https://bitbucket.org/dkessner/artificial_selection_pipeline`

4.5.2 Generation of random trait architectures

In order to investigate how the genetic architecture of a trait affects the behavior of QTL allele frequencies under artificial selection, we developed a method to generate random trait architectures with specified parameters. In particular, we were interested in how the number of QTLs contributing to the trait and the heritability of the trait affect the power to detect QTLs. In addition, we wanted to investigate how the trait architecture affects a focal QTL with a specified effect size and initial allele frequency. Finally, we wanted to ensure that linkage disequilibrium in the simulated starting population is similar to that found in populations used in experimental settings. In the following, we describe our procedure for generating the founding population and trait architecture in our simulations.

Starting with founder individuals for which we have full-genome haplotypes, we run neutral forward simulations with `forqs` to recombine the haplotypes and expand the population size. This results in a mixed population that is 2-3x larger than the desired starting population for the selection experiment. For our founder individuals, we used the publicly available SNP data from 162 Drosophila inbred lines representing Freeze 1 of the Drosophila Genetic Reference Panel (DGRP) project (Mackay *et al.*, 2012). Our procedure is similar to the experimental procedure used by Turner and Miller (2012), where individuals from the DGRP inbred lines are allowed to mate randomly for several generations in order to create a mixed population with genetic variation and linkage disequilibrium similar to the natural popluation from which the inbred lines were derived.

In some cases we additionally specify a focal QTL, for which we specify the locus, effect size and initial allele frequency. We create a starting population by randomly selecting a subset of individuals from the mixed population. We ensure that the focal QTL has the desired allele frequency by choosing individuals in Hardy-Weinberg proportions according to their focal QTL genotype.

From the heritability and total variance parameters that we specify, we calculate a target genetic variance for the trait. We choose the remaining QTL positions uniformly at random across the genome, until we have the specified number of QTLs. We choose effect sizes according to a standard exponential distribution, with random sign for positive/negative effect on the trait. From the individuals' haplotypes and QTL effect sizes, we calculate the current genetic variance of the trait in the population. We then scale the QTL effect sizes so that the genetic variance is equal to the target genetic variance. In the case where we have a focal QTL, we keep the focal QTL effect size constant, and scale the other QTL effect sizes – this requires iteration until the genetic variance is close to the target genetic variance, within a specified tolerance.

4.5.3 Realized selection coefficient

We first recall the deterministic model of selection on a single locus in a diploid population. Suppose a locus has two alleles A_0 and A_1. If p is the frequency of A_1, then the allele frequency change is given by:

$$\Delta p = p(1-p)\frac{w_1^* - w_0^*}{\bar{w}}$$

where w_0^* and w_1^* are the marginal fitnesses of the A_0 and A_1 alleles, respectively, and $\bar{w}$ is the mean fitness of the population (Rice, 2004).

In the special case where A_1 has additive selection coefficient s, this becomes:

$$\Delta p = p(1-p)\frac{s}{1+2ps}$$

We define the *realized selection coefficient* for a given p and Δp to be the selection coefficient s that would result in this allele frequency change in the deterministic case. By solving the above equation for s, we obtain a formula for the realized selection coefficient:

$$s_{\text{realized}}(p, \Delta p) = \frac{\Delta p}{p(1-p) - 2p\Delta p}$$

4.5.4 Calculation of power and false positive rate

We explored several methods for calculating the power and false positive rate associated with the detection of QTLs by allele frequency differences between populations.

The simplest method to measure power is to calculate the proportion of QTLs detected. However, we feel that a more relevant measure of power is the proportion of the genetic variance in the initial population that is explained by the detected QTLs. This measure appropriately gives greater weight to QTLs responsible for more of the genetic variance. Figure 4.11AB illustrates the difference between these two measures of power: at a false positive rate of 10^{-4}, only 50% of the QTLs are detected, but these QTLs are responsible for more than 75% of the genetic variance of the trait.

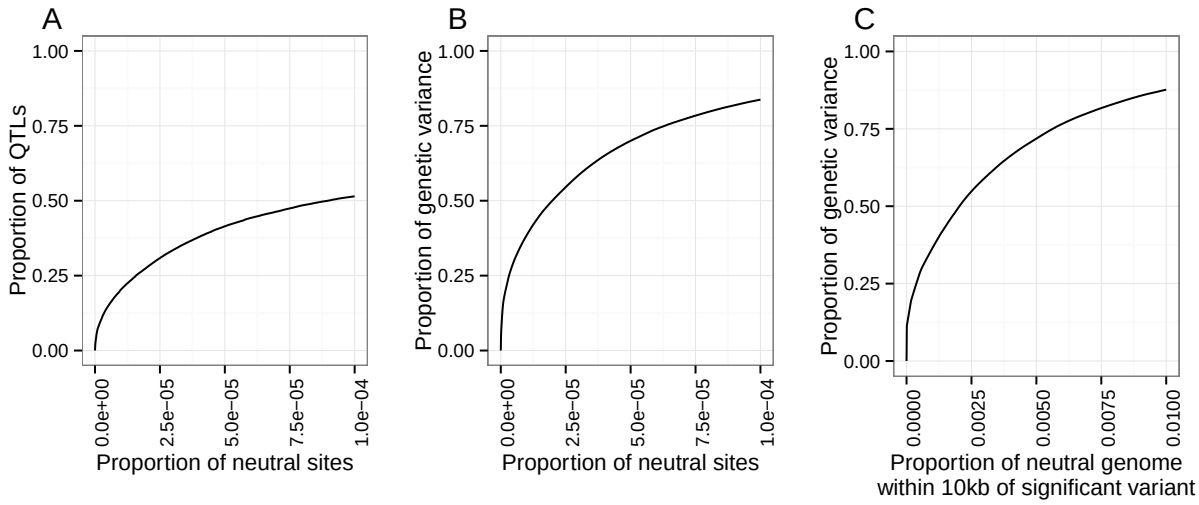

Figure 4.11: **Comparison of three methods for calculating and interpreting power and false positive rate.** A) Power is measured by proportion of QTLs detected, and false positive rate is measured by proportion of neutral variant sites detected. B) Power is measured by the proportion of genetic variance in the founder population explained by the detected QTLs; false positive rate as in A. C) Power as in B; false positive rate is measured by proportion of the neutral genome covered by the detection region. In this case, the ROC curve shows that over 75% of the genetic variance is explained by QTLs in a detection region which covers 1% of the neutral genome.

Furthermore, calculating the false positive rate as the proportion of neutral variants detected does not capture the clustering of detected variants due to linkage disequilibrium. In addition, significantly diverged variants are not necessarily causal, but indicate that the region nearby may contain a causal variant. To address these issues, we define a detection region to include all variants within a specified radius of any variant whose D value exceeds a given threshold (see Results, "Measurement of power to detect and localize QTLs"). This leads to a natural definition of false positive rate: the proportion of the neutral genome covered by the detection region (see Figure 4.9 for an illustration).

We calculated the false positive rate in this way with radii of 10kb, 100kb, and 1mb. We found that using a radius of 100kb or 1mb led to true regions that covered a substantial portion of the genome in the cases where there were a large number of QTLs. Thus, we found that using the 10kb radius for the detection region, together with measuring power as the proportion of variance explained, gave the most interpretable results.

4.5.5 Analysis pipeline

The forward simulator `forqs` efficiently simulates the entire genome of each individual by tracking haplotype chunks. `forqs` outputs data files that represent each chromosome of each individual as a mosaic of founder haplotypes. In order to obtain the neutral variants carried by an individual, the neutral variation on founder chromosomes must be propagated to the individual's mosaic chromosomes.

In our simulation framework, we use two forward simulations. The first simulation creates a mixed population from the founder haplotypes, after which we generate the random trait architecture. The second simulation represents the selection experiment. Individuals in the final populations are mosaics of individuals in the mixed population, which are in turn mosaics of the founders. We implemented a custom program to handle this two-step propagation of neutral variation. Given founder sequences, the mixed population, and the final population, the program calculates the allele frequency in the final population of each variant in the genome. We note that while this step does not require a large amount of computation, it is I/O-intensive because it uses the full DGRP variant data set for the founder sequences, which consists of 162 haplotypes at approximately 5 million loci.

After calculating allele frequencies for each population, we calculate the allele frequency difference D for each high-low population pair under consideration (multiple pairs for the replication analyses). After sorting the D values, we begin with the highest D value and iteratively decrease the threshold to obtain data equivalent to an ROC curve (power and false positive rates)

for that simulation run. We note that this step depends on the method for calculating power and false positive rate, so we performed it once for each method we described above in "Calculation of power and false positive rate".

We obtain average ROC curves by calculating the average power over replicate simulation runs at regularly spaced false positive rates, where the power for a particular run at a given false positive rate is obtained by linear interpolation between points on its ROC curve.

4.5.6 Empirical error distributions

To investigate the power increase due to the use of haplotype-based allele frequency estimates, we needed to simulate pooled sequence reads from a population, followed by haplotype frequency estimation in sliding windows across the genome, using the `harp` method and software detailed in Kessner *et al.* (2013). Because this procedure is computationally expensive, and due to the large number of simulations involved in this study, it was not feasible to do this for each simulated experiment.

As an alternative, we obtained empirical error distributions, which we later used to add random errors to true allele frequencies. Because errors in the haplotype frequency estimation depend on the length scale of recombination, we ran replicate neutral simulations for varying numbers of generations (from 40 to 400). Haplotypes surrounding selected QTLs are expected to be longer than in neutral regions, so our empirical error distributions are conservative. We simulated pooled sequence reads from the final populations using a new program `simreads_forqs` based on the read simulator used in Kessner *et al.* (2013). Haplotype frequency estimation was performed with `harp` in overlapping sliding 200kb windows within a single 1mb region. We then calculated allele frequencies at variant sites within the region using read counts. By considering allele frequencies in bins of size .05, we obtained a frequency-dependent empirical error distribution. Similarly, we also derived allele frequency estimates from the local haplotype frequencies, from which we obtained frequency-dependent empirical error distributions for each

generation count. We found that after ~ 200 generations, the haplotype-based estimates were no better than the read-count-based estimates; this behavior is expected as the recombination length scale approaches the window size used for haplotype frequency estimation (200kb in this analysis). Hence, for this analysis we only considered experimental scenarios lasting less than 200 generations.

APPENDIX A

Additional analysis of haplotype frequency estimation in metagenomics applications

A typical metagenomics analysis includes an estimate of the relative abundances of the various taxa comprising the microbial community. In this Appendix, I compare relative abundance estimates obtained by the maximum-likelihood method of Chapter 2 (implemented in the `harp` software) to those calculated by the commonly used metagenomics analysis pipeline `QIIME` (Quantitative Insights Into Microbial Ecology, `http://qiime.org`). In particular, for the scenario where known reference sequences are used for the estimation, I show that `harp` produces better relative abundance estimates at the species level, as well as at higher taxonomic levels. In addition, I show that the improved relative abundance estimates lead to better estimates of community diversity.

A.1 Mock community analysis

We first considered a scenario based on Mock Community samples used in the Human Microbiome Project (HMP, `http://www.hmpdacc.org`). The Mock Community samples contain 21 species at known abundances. We simulated 75bp single-end reads generated from 16S rDNA, at high coverage (1000x), using base quality score distributions obtained from the publicly available HMP data sets.

For the `harp` analysis, we first aligned all reads to each reference sequence using the `bwa`

program (Li and Durbin, 2010). For the QIIME analysis, we calculated relative abundances using the "closed reference" option. With this option specified, the QIIME pipeline performs a read alignment, followed by assignment of each read to the best matching reference sequence. Finally, QIIME calculates relative abundances based on the proportion of reads assigned to each taxon. By default, QIIME requires reads to match a reference sequence with 97% sequence similarity. For comparison, we also used sequence similarity parameters of 90% and 80%.

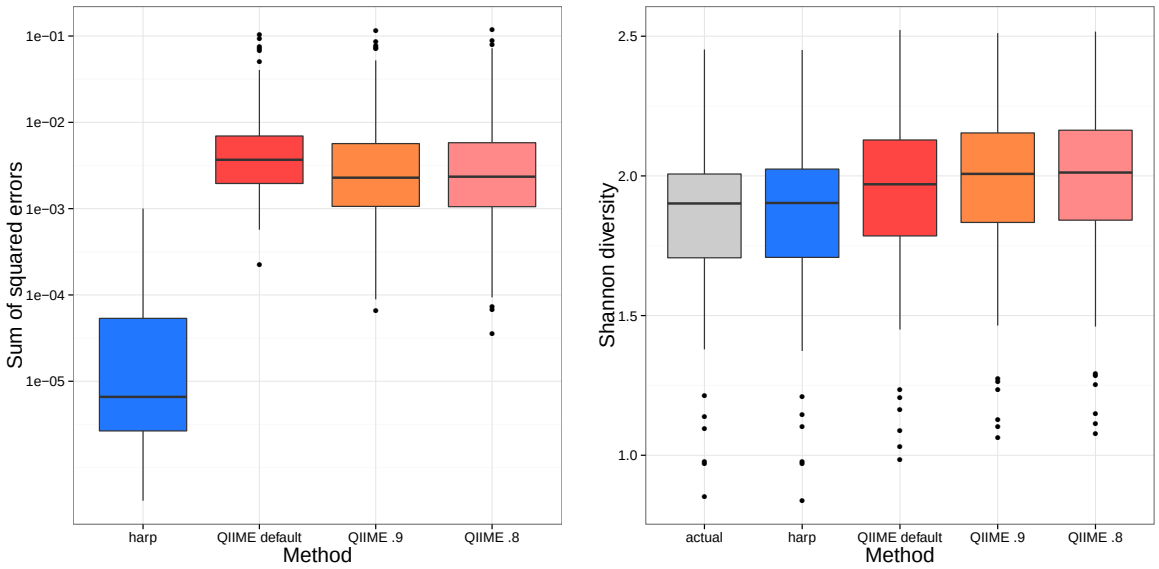

Figure A.1: **Mock community comparison of harp to QIIME.** harp produces better species relative abundance estimates than QIIME (left), leading to better estimates of diversity (right). [Simulations of 75bp reads from a pool of 21 species at 1000x pooled coverage]

Similar to the results in Chapter 2, we found that harp estimates showed an improvement over those obtained from QIIME, both for species relative abundance and for species diversity (measured by the Shannon diversity index) (Figure A.1). We additionally found that the estimates of relative abundance at higher taxonomic levels also improved (Figure A.2).

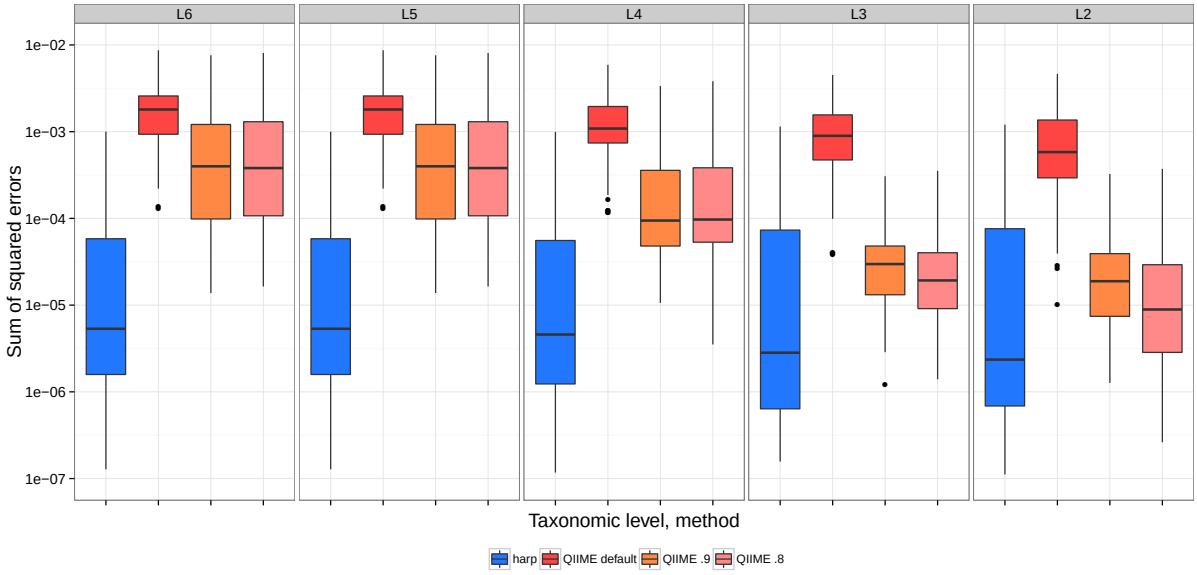

Figure A.2: **Mock community comparison of `harp` to `QIIME` – higher level taxa.** `harp` produces better relative abundance estimates than `QIIME` at higher taxonomic levels. [Simulations of 75bp reads from a pool of 21 species at 1000x pooled coverage; L6=genus, L5=family, L4=order, L3=class, L2=phylum]

A.2 Larger community analysis

We next simulated a larger microbial community, with each simulation drawing 200 species randomly from a list of 1000 known species, at frequencies drawn from a symmetric Dirichlet distribution (common parameter .2). We then simulated high coverage (1000x) 75bp reads from the sample, and analyzed with `harp` and `QIIME` with default parameters, using a closed reference set consisting of the 1000 known species.

As in the mock community scenario above, `harp` showed significant improvement in performance over read-count-based estimates of relative abundance, both at the species level (Figure A.3) and at higher taxonomic levels (Figure A.4). In addition, community diversity estimates improved as a result (Figures A.3 and A.5).

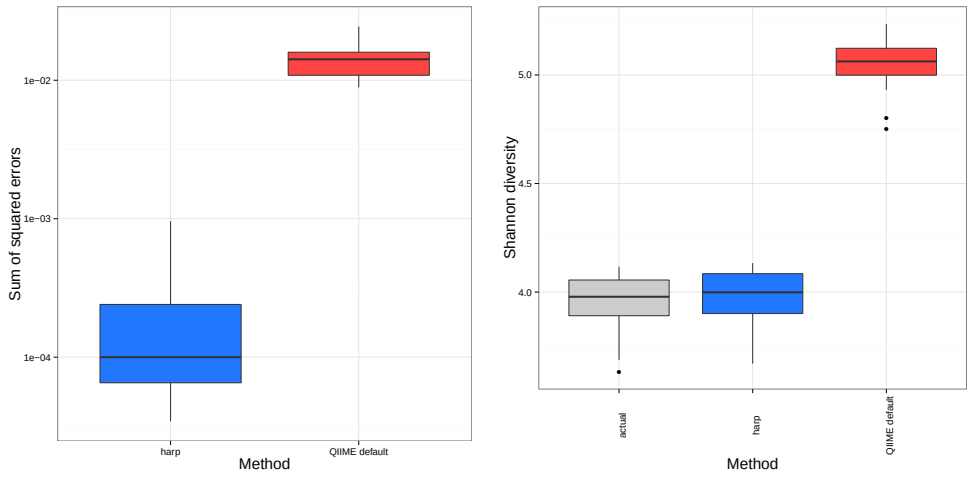

Figure A.3: **Larger community comparison of `harp` to `QIIME`.** `harp` produces better species relative abundance estimates than `QIIME` (left), leading to better estimates of diversity (right). [Simulations of 75bp reads from a pool of 200 species at 1000x pooled coverage]

A.3 Discussion

This short analysis shows that estimates of the relative abundances of taxa in a mixed sample can be improved by the maximum likelihood method of Chapter 2, whether these taxa are distinguished at the species level or a higher taxonomic level. The improvement comes from the incorporation of base quality scores into the likelihood calculation and from combining the information from reads across the sequenced region (16S in this case). As a result, estimates of community diversity are also improved.

We note that in this analysis we have assumed that we have a set of reference sequences that includes all the constituent species in the pool. Before running `harp` to compute relative abundances, the raw sequence reads must first be mapped to each reference sequence to obtain a haplotype likelihood for each reference species. This poses a potential performance issue as the number of reference sequences increases. One possible approach to handle this issue is to map reads to multiple references simultaneously, where the reference sequences are stored in

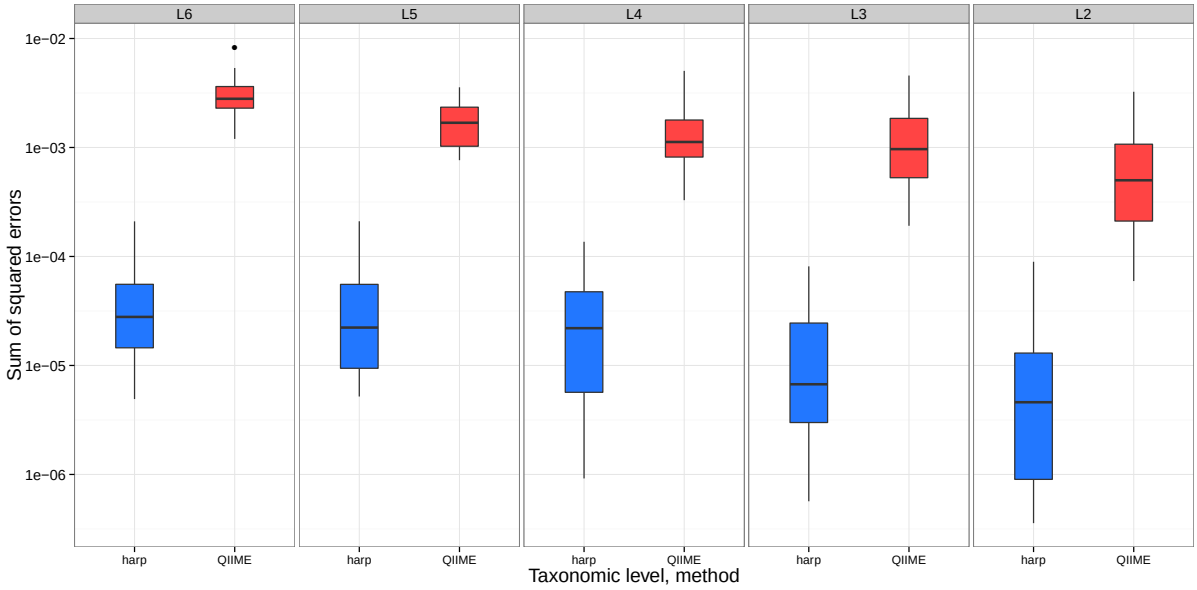

Figure A.4: **Larger community comparison of `harp` to `QIIME`.** `harp` produces better relative abundance estimates than `QIIME` at higher taxonomic levels. [Simulations of 75bp reads from a pool of 200 species at 1000x pooled coverage; L6=genus, L5=family, L4=order, L3=class, L2=phylum]

a tree structure that reflects the phylogenetic relationships between the reference species. This approach would speed up read mapping by, for example, avoiding attempts to map a read to a particular locus in groups of closely related species once it has been established that the read does not map to that locus in one representative species.

Further discussion of the use of this method in the metagenomics context can be found in Chapter 2.

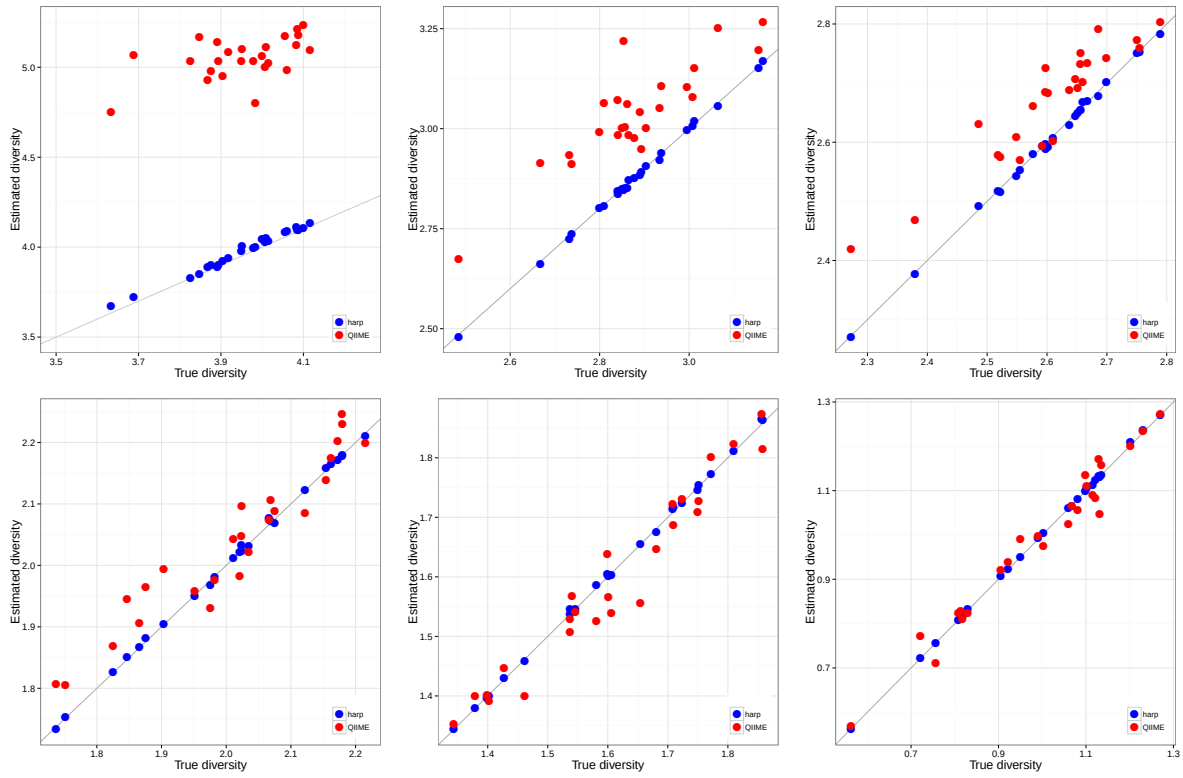

Figure A.5: **Estimated vs. true diversity.** Shown are comparisons of diversity estimates of `harp` and `QIIME` vs. true diversity, at different taxon levels. (top row) species, genus, family (bottom row) order, class, phylum [Simulations of 75bp reads from a pool of 200 species at 1000x pooled coverage]

BIBLIOGRAPHY

Aberer, A. J. and Stamatakis, A. (2013). Rapid forward-in-time simulation at the chromosome and genome level. *BMC Bioinformatics*, **14**, 216.

Baldwin-Brown, J. G., Long, A. D., and Thornton, K. R. (2014). The power to detect quantitative trait Loci using resequenced, experimentally evolved populations of diploid, sexual organisms. *Mol. Biol. Evol.*, **31**(4), 1040–1055.

Burke, M. K., Dunham, J. P., Shahrestani, P., Thornton, K. R., Rose, M. R., and Long, A. D. (2010). Genome-wide analysis of a long-term evolution experiment with Drosophila. *Nature*, **467**(7315), 587–90.

Carvajal-Rodriguez, A. (2008). GENOMEPOP: a program to simulate genomes in populations. *BMC Bioinformatics*, **9**, 223.

Chadeau-Hyam, M., Hoggart, C. J., O'Reilly, P. F., Whittaker, J. C., De Iorio, M., and Balding, D. J. (2008). Fregene: simulation of realistic sequence-level data in populations and ascertained samples. *BMC Bioinformatics*, **9**, 364.

Cheesman, S. J., de Roode, J. C., Read, A. F., and Carter, R. (2003). Real-time quantitative PCR for analysis of genetically mixed infections of malaria parasites: technique validation and applications. *Molecular and Biochemical Parasitology*, **131**(2), 83–91.

Chen, G. K., Marjoram, P., and Wall, J. D. (2009). Fast and flexible simulation of DNA sequence data. *Genome Res.*, **19**(1), 136–142.

Comeron, J. M., Ratnappan, R., and Bailin, S. (2012). The many landscapes of recombination in Drosophila melanogaster. *PLoS Genet.*, **8**(10), e1002905.

Crow, J. F. (2007). Haldane, Bailey, Taylor and recombinant-inbred lines. *Genetics*, **176**(2), 729–732.

Cutler, D. J. and Jensen, J. D. (2010). To pool, or not to pool? *Genetics*, **186**(1), 41–3.

Darvasi, A. and Soller, M. (1995). Advanced intercross lines, an experimental population for fine genetic mapping. *Genetics*, **141**(3), 1199–1207.

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, **39**(1), 1–38.

DeSantis, T. Z., Hugenholtz, P., Larsen, N., Rojas, M., Brodie, E. L., Keller, K., Huber, T., Dalevi, D., Hu, P., and Andersen, G. L. (2006). Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Applied and Environmental Microbiology*, **72**(7), 5069–72.

Earley, E. J. and Jones, C. D. (2011). Next-generation mapping of complex traits with phenotype-based selection and introgression. *Genetics*, **189**(4), 1203–9.

Ehrenreich, I. M., Torabi, N., Jia, Y., Kent, J., Martis, S., Shapiro, J. A., Gresham, D., Caudy, A. A., and Kruglyak, L. (2010). Dissection of genetically complex traits with extremely large pools of yeast segregants. *Nature*, **464**(7291), 1039–1042.

Ewens, W. J. (2004). *Mathematical Population Genetics 1: Theoretical Introduction (Interdisciplinary Applied Mathematics) (v. 1)*. Springer, 2nd edition.

Ewing, G. and Hermisson, J. (2010). MSMS: a coalescent simulation program including recombination, demographic structure and selection at a single locus. *Bioinformatics*, **26**(16), 2064–2065.

Excoffier, L. and Foll, M. (2011). fastsimcoal: a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios. *Bioinformatics*, **27**(9), 1332–1334.

Excoffier, L., Dupanloup, I., Huerta-Sanchez, E., Sousa, V. C., and Foll, M. (2013). Robust demographic inference from genomic and SNP data. *PLoS Genet.*, **9**(10), e1003905.

Falconer, D. S. (1992). Early selection experiments. *Annu. Rev. Genet.*, **26**, 1–14.

Falconer, D. S. and Mackay, T. F. (1996). *Introduction to Quantitative Genetics (4th Edition)*. Benjamin Cummings, 4 edition.

Felsenstein, J. (1974). The evolutionary advantage of recombination. *Genetics*, **78**(2), 737–756.

Felsenstein, J. (1987). Sex and the evolution of recombination. In R. E. Michod and B. R. Levin, editors, *The Evolution of Sex: An Examination of Current Ideas*, pages 74–86. Sinauer Associates Inc, Sunderland, MA.

Fu, Y. X. (1995). Statistical properties of segregating sites. *Theor Popul Biol*, **48**(2), 172–197.

Futschik, A. and Schlotterer, C. (2010). The next generation of molecular markers from massively parallel sequencing of pooled DNA samples. *Genetics*, **186**(1), 207–218.

Garland, T. and Rose, M. R., editors (2009). *Experimental Evolution: Concepts, Methods, and Applications of Selection Experiments*. University of California Press, 0 edition.

Gasbarra, D., Kulathinal, S., Pirinen, M., and Sillanpää, M. J. (2009). Estimating haplotype frequencies by combining data from large DNA pools with database information. *IEEE/ACM Transactions on Computational Biology and Bioinformatics / IEEE, ACM*, **8**(1), 36–44.

Gillespie, J. H. (2004). *Population Genetics: A Concise Guide*. Johns Hopkins University Press, 2nd edition.

Grossman, S. R., Shlyakhter, I., Shylakhter, I., Karlsson, E. K., Byrne, E. H., Morales, S., Frieden, G., Hostetter, E., Angelino, E., Garber, M., Zuk, O., Lander, E. S., Schaffner, S. F., and Sabeti, P. C. (2010). A composite of multiple signals distinguishes causal variants in regions of positive selection. *Science*, **327**(5967), 883–886.

Haiminen, N., Utro, F., Lebreton, C., Flament, P., Karaman, Z., and Parida, L. (2013). Efficient in silico chromosomal representation of populations via indexing ancestral genomes. *Algorithms*, **6**(3), 430–441.

Hastings, I. M. and Smith, T. a. (2008). MalHaploFreq: a computer programme for estimating malaria haplotype frequencies from blood samples. *Malaria Journal*, **7**, 130.

Hastings, I. M., Nsanzabana, C., and Smith, T. a. (2010). A comparison of methods to detect and quantify the markers of antimalarial drug resistance. *The American Journal of Tropical Medicine and Hygiene*, **83**(3), 489–95.

Hernandez, R. D. (2008). A flexible forward simulator for populations subject to selection and demography. *Bioinformatics*, **24**(23), 2786–2787.

Hill, W. G. and Robertson, A. (1966). The effect of linkage on limits to artificial selection. *Genet. Res.*, **8**(3), 269–294.

Hoban, S., Bertorelle, G., and Gaggiotti, O. E. (2011). Computer simulations: tools for population and evolutionary genetics. *Nat. Rev. Genet.*, **13**(2), 110–122.

Huang, W., Richards, S., Carbone, M. a., Zhu, D., Anholt, R. R. H., Ayroles, J. F., Duncan, L., Jordan, K. W., Lawrence, F., Magwire, M. M., Warner, C. B., Blankenburg, K., Han, Y., Javaid, M., Jayaseelan, J., Jhangiani, S. N., Muzny, D., Ongeri, F., Perales, L., Wu, Y.-Q., Zhang, Y., Zou, X., Stone, E. a., Gibbs, R. a., and Mackay, T. F. C. (2012). Epistasis dominates the genetic architecture of Drosophila quantitative traits. *Proceedings of the National Academy of Sciences*, pages 1–7.

Hudson, R. R. (2002). Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, **18**(2), 337–338.

Hudson, R. R. and Kaplan, N. L. (1988). The coalescent process in models with selection and recombination. *Genetics*, **120**(3), 831–840.

Human Microbiome Project Consortium (2012). Structure, function and diversity of the healthy human microbiome. *Nature*, **486**(7402), 207–14.

Hunt, P., Fawcett, R., Carter, R., and Walliker, D. (2005). Estimating SNP proportions in populations of malaria parasites by sequencing: validation and applications. *Molecular and Biochemical Parasitology*, **143**(2), 173–82.

International Human Genome Sequencing Consortium, Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., Funke, R., Gage, D., Harris, K., Heaford, A., Howland, J., Kann, L., Lehoczky, J., LeVine, R., McEwan, P., McKernan, K., Meldrim, J., Mesirov, J. P., Miranda, C., Morris, W., Naylor, J., Raymond, C., Rosetti, M., Santos, R., Sheridan, A., Sougnez, C., Stange-Thomann, N., Stojanovic, N., Subramanian, A., Wyman, D., Rogers, J., Sulston, J., Ainscough, R., Beck, S., Bentley, D., Burton, J., Clee, C., Carter, N., Coulson, A., Deadman, R., Deloukas, P., Dunham, A., Dunham, I., Durbin, R., French, L., Grafham, D., Gregory, S., Hubbard, T., Humphray, S., Hunt, A., Jones, M., Lloyd, C., McMurray, A., Matthews, L., Mercer, S., Milne, S., Mullikin, J. C., Mungall, A., Plumb, R., Ross, M., Shownkeen, R., Sims, S., Waterston, R. H., Wilson, R. K., Hillier, L. W., McPherson, J. D., Marra, M. A., Mardis, E. R., Fulton, L. A., Chinwalla, A. T., Pepin, K. H., Gish, W. R., Chissoe, S. L., Wendl, M. C., Delehaunty, K. D., Miner, T. L., Delehaunty, A., Kramer, J. B., Cook, L. L., Fulton, R. S., Johnson, D. L., Minx, P. J., Clifton, S. W., Hawkins, T., Branscomb, E., Predki, P., Richardson, P., Wenning, S., Slezak, T., Doggett, N., Cheng, J. F., Olsen, A., Lucas, S., Elkin, C., Uberbacher, E., Frazier, M., Gibbs, R. A., Muzny, D. M., Scherer, S. E., Bouck, J. B., Sodergren, E. J., Worley, K. C., Rives, C. M., Gorrell, J. H., Metzker, M. L., Naylor, S. L., Kucherlapati, R. S., Nelson, D. L., Weinstock, G. M., Sakaki, Y., Fujiyama, A., Hattori, M., Yada, T., Toyoda, A., Itoh, T., Kawagoe, C., Watanabe, H., Totoki, Y., Taylor, T., Weissenbach, J., Heilig, R., Saurin, W., Artiguenave, F., Brottier, P., Bruls, T., Pelletier, E., Robert, C., Wincker, P., Smith, D. R., Doucette-Stamm, L., Rubenfield, M., Weinstock, K., Lee, H. M., Dubois, J., Rosenthal, A.,

Platzer, M., Nyakatura, G., Taudien, S., Rump, A., Yang, H., Yu, J., Wang, J., Huang, G., Gu, J., Hood, L., Rowen, L., Madan, A., Qin, S., Davis, R. W., Federspiel, N. A., Abola, A. P., Proctor, M. J., Myers, R. M., Schmutz, J., Dickson, M., Grimwood, J., Cox, D. R., Olson, M. V., Kaul, R., Raymond, C., Shimizu, N., Kawasaki, K., Minoshima, S., Evans, G. A., Athanasiou, M., Schultz, R., Roe, B. A., Chen, F., Pan, H., Ramser, J., Lehrach, H., Reinhardt, R., McCombie, W. R., de la Bastide, M., Dedhia, N., Blocker, H., Hornischer, K., Nordsiek, G., Agarwala, R., Aravind, L., Bailey, J. A., Bateman, A., Batzoglou, S., Birney, E., Bork, P., Brown, D. G., Burge, C. B., Cerutti, L., Chen, H. C., Church, D., Clamp, M., Copley, R. R., Doerks, T., Eddy, S. R., Eichler, E. E., Furey, T. S., Galagan, J., Gilbert, J. G., Harmon, C., Hayashizaki, Y., Haussler, D., Hermjakob, H., Hokamp, K., Jang, W., Johnson, L. S., Jones, T. A., Kasif, S., Kaspryzk, A., Kennedy, S., Kent, W. J., Kitts, P., Koonin, E. V., Korf, I., Kulp, D., Lancet, D., Lowe, T. M., McLysaght, A., Mikkelsen, T., Moran, J. V., Mulder, N., Pollara, V. J., Ponting, C. P., Schuler, G., Schultz, J., Slater, G., Smit, A. F., Stupka, E., Szustakowski, J., Thierry-Mieg, D., Thierry-Mieg, J., Wagner, L., Wallis, J., Wheeler, R., Williams, A., Wolf, Y. I., Wolfe, K. H., Yang, S. P., Yeh, R. F., Collins, F., Guyer, M. S., Peterson, J., Felsenfeld, A., Wetterstrand, K. A., Patrinos, A., Morgan, M. J., de Jong, P., Catanese, J. J., Osoegawa, K., Shizuya, H., Choi, S., Chen, Y. J., and Szustakowki, J. (2001). Initial sequencing and analysis of the human genome. *Nature*, **409**(6822), 860–921.

Ito, T., Chiku, S., Inoue, E., Tomita, M., Morisaki, T., Morisaki, H., and Kamatani, N. (2003). Estimation of Haplotype Frequencies, Linkage-Disequilibrium Measures, and Combination of Haplotype Copies in Each Pool by Use of Pooled DNA Data. *American Journal of Human Genetics*, (2001), 384–398.

Johansson, A. M., Pettersson, M. E., Siegel, P. B., and Carlborg, O. (2010). Genome-wide effects of long-term divergent selection. *PLoS Genet.*, **6**(11), e1001188.

Keightley, P. D. and Bulfield, G. (1993). Detection of quantitative trait loci from frequency changes of marker alleles under selection. *Genetical Research*, **62**(3), 195–203.

Kessner, D. and Novembre, J. (2014). forqs: forward-in-time simulation of recombination, quantitative traits and selection. *Bioinformatics*, **30**(4), 576–577.

Kessner, D., Turner, T. L., and Novembre, J. (2013). Maximum likelihood estimation of frequencies of known haplotypes from pooled sequence data. *Mol. Biol. Evol.*, **30**(5), 1145–1158.

Kim, Y. and Stephan, W. (1999). Allele frequency changes in artificial selection experiments: statistical power and precision of QTL mapping. *Genet. Res.*, **73**, 177–184.

Kirkpatrick, B., Armendariz, C. S., Karp, R. M., and Halperin, E. (2007). HAPLOPOOL: improving haplotype frequency estimation through DNA pools and phylogenetic modeling. *Bioinformatics*, **23**(22), 3048–55.

Kofler, R. and Schlotterer, C. (2014). A guide for the design of evolve and resequencing studies. *Mol. Biol. Evol.*, **31**(2), 474–483.

Kofler, R., Orozco-terWengel, P., De Maio, N., Pandey, R. V., Nolte, V., Futschik, A., Kosiol, C., and Schlötterer, C. (2011). PoPoolation: a toolbox for population genetic analysis of next generation sequencing data from pooled individuals. *PLOS ONE*, **6**(1), e15925.

Korte, A. and Farlow, A. (2013). The advantages and limitations of trait analysis with GWAS: a review. *Plant Methods*, **9**(1), 29.

Kuk, A. Y. C., Zhang, H., and Yang, Y. (2009). Computationally feasible estimation of haplotype frequencies from pooled DNA with and without Hardy-Weinberg equilibrium. *Bioinformatics*, **25**(3), 379–86.

Lai, C. Q., Leips, J., Zou, W., Roberts, J. F., Wollenberg, K. R., Parnell, L. D., Zeng, Z. B., Ordovas, J. M., and Mackay, T. F. (2007). Speed-mapping quantitative trait loci using microarrays. *Nat. Methods*, **4**(10), 839–841.

Lambert, B. W., Terwilliger, J. D., and Weiss, K. M. (2008). ForSim: a tool for exploring the genetic architecture of complex traits with controlled truth. *Bioinformatics*, **24**(16), 1821–1822.

Lange, K. (2010). *Numerical Analysis for Statisticians (Statistics and Computing)*. Springer, 2nd ed. edition.

Laurie, C. C., Chasalow, S. D., LeDeaux, J. R., McCarroll, R., Bush, D., Hauge, B., Lai, C., Clark, D., Rocheford, T. R., and Dudley, J. W. (2004). The genetic architecture of response to long-term artificial selection for oil concentration in the maize kernel. *Genetics*, **168**(4), 2141–2155.

Ley, R., Turnbaugh, P. J., Klein, S., and Gordon, J. I. (2006). Human gut microbes associated with obesity. *Nature*, **444**.

Li, H. and Durbin, R. (2010). Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, **26**(5), 589–95.

Li, H. and Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Brief. Bioinformatics*, **11**(5), 473–483.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**(16), 2078–9.

Li, X., Foulkes, A. S., Yucel, R. M., and Rich, S. M. (2007). An expectation maximization approach to estimate malaria haplotype frequencies in multiply infected children. *Statistical Applications in Genetics and Molecular Biology*, **6**(1), Article33.

Long, Q., Jeffares, D. C., Zhang, Q., Ye, K., Nizhynska, V., Ning, Z., Tyler-Smith, C., and Nordborg, M. (2011). PoolHap: inferring haplotype frequencies from pooled samples by next generation sequencing. *PLOS ONE*, **6**(1), e15292.

Mackay, T. F. (2001). Quantitative trait loci in Drosophila. *Nat. Rev. Genet.*, **2**(1), 11–20.

Mackay, T. F., Stone, E. A., and Ayroles, J. F. (2009). The genetics of quantitative traits: challenges and prospects. *Nat. Rev. Genet.*, **10**(8), 565–577.

Mackay, T. F. C., Richards, S., Stone, E. a., Barbadilla, A., Ayroles, J. F., Zhu, D., Casillas, S., Han, Y., Magwire, M. M., Cridland, J. M., Richardson, M. F., Anholt, R. R. H., Barrón, M., Bess, C., Blankenburg, K. P., Carbone, M. A., Castellano, D., Chaboub, L., Duncan, L., Harris, Z., Javaid, M., Jayaseelan, J. C., Jhangiani, S. N., Jordan, K. W., Lara, F., Lawrence, F., Lee, S. L., Librado, P., Linheiro, R. S., Lyman, R. F., Mackey, A. J., Munidasa, M., Muzny, D. M., Nazareth, L., Newsham, I., Perales, L., Pu, L.-L., Qu, C., Ràmia, M., Reid, J. G., Rollmann, S. M., Rozas, J., Saada, N., Turlapati, L., Worley, K. C., Wu, Y.-Q., Yamamoto, A., Zhu, Y., Bergman, C. M., Thornton, K. R., Mittelman, D., and Gibbs, R. a. (2012). The Drosophila melanogaster Genetic Reference Panel. *Nature*, **482**(7384), 173–178.

Messer, P. W. (2013). SLiM: simulating evolution with selection and linkage. *Genetics*, **194**(4), 1037–1039.

Metzker, M. L. (2010). Sequencing technologies - the next generation. *Nat. Rev. Genet.*, **11**(1), 31–46.

Mizrahi-Man, O., Davenport, E. R., and Gilad, Y. (2013). Taxonomic Classification of Bacterial 16S rRNA Genes Using Short Sequencing Reads: Evaluation of Effective Study Designs. *PLoS ONE*, **8**(1), e53608.

Neuenschwander, S., Hospital, F., Guillaume, F., and Goudet, J. (2008). quantiNemo: an individual-based program to simulate quantitative traits with explicit genetic architecture in a dynamic metapopulation. *Bioinformatics*, **24**(13), 1552–1553.

NIH HMP Working Group (2009). The NIH Human Microbiome Project. *Genome Research*, pages 2317–2323.

Niu, T. (2004). Algorithms for inferring haplotypes. *Genetic Epidemiology*, **27**(4), 334–47.

Nuzhdin, S. V., Harshman, L. G., Zhou, M., and Harmon, K. (2007). Genome-enabled hitchhiking mapping identifies QTLs for stress resistance in natural Drosophila. *Heredity*, **99**(3), 313–21.

O'Fallon, B. (2010). TreesimJ: a flexible, forward time population genetic simulator. *Bioinformatics*, **26**(17), 2200–2201.

Orozco-terWengel, P., Kapun, M., Nolte, V., Kofler, R., Flatt, T., and Schlotterer, C. (2012). Adaptation of Drosophila to a novel laboratory environment reveals temporally heterogeneous trajectories of selected alleles. *Mol. Ecol.*, **21**(20), 4931–4941.

Padhukasahasram, B., Marjoram, P., Wall, J. D., Bustamante, C. D., and Nordborg, M. (2008). Exploring population genetic models with recombination using efficient forward-time simulations. *Genetics*, **178**(4), 2417–2427.

Parts, L., Cubillos, F. a., Warringer, J., Jain, K., Salinas, F., Bumpstead, S. J., Molin, M., Zia, A., Simpson, J. T., Quail, M. a., Moses, A., Louis, E. J., Durbin, R., and Liti, G. (2011). Revealing the genetic structure of a trait by sequencing a population under selection. *Genome Research*, **21**(7), 1131–8.

Pe'er, I. and Beckmann, J. S. (2003). Resolution of haplotypes and haplotype frequencies from SNP genotypes of pooled samples. *Proceedings of the Seventh Annual International Conference on Computational Molecular Biology - RECOMB '03*, pages 237–246.

Peng, B. and Kimmel, M. (2005). simuPOP: a forward-time population genetics simulation environment. *Bioinformatics*, **21**(18), 3686–3687.

Pirinen, M. (2009). Estimating population haplotype frequencies from pooled SNP data using incomplete database information. *Bioinformatics*, **25**(24), 3296–302.

Remolina, S. C., Chang, P. L., Leips, J., Nuzhdin, S. V., and Hughes, K. A. (2012). Genomic basis of aging and life-history evolution in Drosophila melanogaster. *Evolution*, **66**(11), 3390–3403.

Rice, S. H. (2004). *Evolutionary Theory*. Sinauer Associates, Inc.

Sabeti, P. C., Varilly, P., Fry, B., Lohmueller, J., Hostetter, E., Cotsapas, C., Xie, X., Byrne, E. H., McCarroll, S. a., Gaudet, R., Schaffner, S. F., Lander, E. S., Frazer, K. a., Ballinger, D. G., Cox, D. R., Hinds, D. a., Stuve, L. L., Gibbs, R. a., Belmont, J. W., Boudreau, A., Hardenbol, P., Leal, S. M., Pasternak, S., Wheeler, D. a., Willis, T. D., Yu, F., Yang, H., Zeng, C., Gao, Y., Hu, H., Hu, W., Li, C., Lin, W., Liu, S., Pan, H., Tang, X., Wang, J., Wang, W., Yu, J., Zhang, B., Zhang, Q., Zhao, H., Zhao, H., Zhou, J., Gabriel, S. B., Barry, R., Blumenstiel, B., Camargo, A., Defelice, M., Faggart, M., Goyette, M., Gupta, S., Moore, J., Nguyen, H., Onofrio, R. C., Parkin, M., Roy, J., Stahl, E., Winchester, E., Ziaugra, L., Altshuler, D., Shen, Y., Yao, Z., Huang, W., Chu, X., He, Y., Jin, L., Liu, Y., Shen, Y., Sun, W., Wang, H., Wang, Y., Wang, Y., Xiong, X., Xu, L., Waye, M. M. Y., Tsui, S. K. W., Xue, H., Wong, J. T.-F., Galver, L. M., Fan, J.-B., Gunderson, K., Murray, S. S., Oliphant, A. R., Chee, M. S., Montpetit, A., Chagnon, F., Ferretti, V., Leboeuf, M., Olivier, J.-F., Phillips, M. S., Roumy, S., Sallée, C., Verner, A., Hudson, T. J., Kwok, P.-Y., Cai, D., Koboldt, D. C., Miller, R. D., Pawlikowska, L., Taillon-Miller, P., Xiao, M., Tsui, L.-C., Mak, W., Song, Y. Q., Tam, P. K. H., Nakamura, Y., Kawaguchi, T., Kitamoto, T., Morizono, T., Nagashima, A., Ohnishi, Y., Sekine, A., Tanaka, T., Tsunoda, T., Deloukas, P., Bird, C. P., Delgado, M., Dermitzakis, E. T., Gwilliam, R., Hunt, S., Morrison, J., Powell, D., Stranger, B. E., Whittaker, P., Bentley, D. R., Daly, M. J., de Bakker, P. I. W., Barrett, J., Chretien, Y. R., Maller, J., McCarroll, S., Patterson, N., Pe'er, I., Price, A., Purcell, S., Richter, D. J., Sabeti, P., Saxena, R., Sham, P. C., Stein, L. D., Krishnan, L., Smith, A. V., Tello-Ruiz, M. K., Thorisson, G. a., Chakravarti, A., Chen, P. E., Cutler, D. J., Kashuk, C. S., Lin, S., Abecasis, G. R., Guan, W., Li, Y., Munro, H. M., Qin, Z. S., Thomas, D. J., McVean, G., Auton, A., Bottolo, L., Cardin, N., Eyheramendy,

S., Freeman, C., Marchini, J., Myers, S., Spencer, C., Stephens, M., Donnelly, P., Cardon, L. R., Clarke, G., Evans, D. M., Morris, A. P., Weir, B. S., Johnson, T. a., Mullikin, J. C., Sherry, S. T., Feolo, M., Skol, A., Zhang, H., Matsuda, I., Fukushima, Y., Macer, D. R., Suda, E., Rotimi, C. N., Adebamowo, C. a., Ajayi, I., Aniagwu, T., Marshall, P. a., Nkwodimmah, C., Royal, C. D. M., Leppert, M. F., Dixon, M., Peiffer, A., Qiu, R., Kent, A., Kato, K., Niikawa, N., Adewole, I. F., Knoppers, B. M., Foster, M. W., Clayton, E. W., Watkin, J., Muzny, D., Nazareth, L., Sodergren, E., Weinstock, G. M., Yakub, I., Birren, B. W., Wilson, R. K., Fulton, L. L., Rogers, J., Burton, J., Carter, N. P., Clee, C. M., Griffiths, M., Jones, M. C., McLay, K., Plumb, R. W., Ross, M. T., Sims, S. K., Willey, D. L., Chen, Z., Han, H., Kang, L., Godbout, M., Wallenburg, J. C., L'Archevêque, P., Bellemare, G., Saeki, K., Wang, H., An, D., Fu, H., Li, Q., Wang, Z., Wang, R., Holden, A. L., Brooks, L. D., McEwen, J. E., Guyer, M. S., Wang, V. O., Peterson, J. L., Shi, M., Spiegel, J., Sung, L. M., Zacharia, L. F., Collins, F. S., Kennedy, K., Jamieson, R., and Stewart, J. (2007). Genome-wide detection and characterization of positive selection in human populations. *Nature*, **449**(7164), 913–8.

Takala, S. L., Smith, D. L., Stine, O. C., Coulibaly, D., Thera, M. a., Doumbo, O. K., and Plowe, C. V. (2006). A high-throughput method for quantifying alleles and haplotypes of the malaria vaccine candidate Plasmodium falciparum merozoite surface protein-1 19 kDa. *Malaria Journal*, **5**, 31.

Teotonio, H., Chelo, I. M., Bradi?, M., Rose, M. R., and Long, A. D. (2009). Experimental evolution reveals natural selection on standing genetic variation. *Nat. Genet.*, **41**(2), 251–257.

Turner, T. L. and Miller, P. M. (2012). Investigating natural variation in Drosophila courtship song by the evolve and resequence approach. *Genetics*, **191**(2), 633–42.

Turner, T. L., Stewart, A. D., Fields, A. T., Rice, W. R., and Tarone, A. M. (2011). Population-based resequencing of experimentally evolved populations reveals the genetic basis of body size variation in Drosophila melanogaster. *PLOS Genetics*, **7**(3), e1001336.

Utsunomiya, Y. T., Perez O'Brien, A. M., Sonstegard, T. S., Van Tassell, C. P., do Carmo, A. S., Meszaros, G., Solkner, J., and Garcia, J. F. (2013). Detecting loci under recent positive selection in dairy and beef cattle by combining different genome-wide scan methods. *PLoS ONE*, **8**(5), e64280.

Voight, B. F., Kudaravalli, S., Wen, X., and Pritchard, J. K. (2006). A map of recent positive selection in the human genome. *PLOS Biology*, **4**(3), e72.

Wakeley, J. and Takahashi, T. (2003). Gene genealogies when the sample size exceeds the effective size of the population. *Mol. Biol. Evol.*, **20**(2), 208–213.

Wang, S., Kidd, K. K., and Zhao, H. (2003). On the Use of DNA Pooling to Estimate Haplotype Frequencies. *Genetic Epidemiology*, **24**, 74–82.

Wegmann, D., Kessner, D. E., Veeramah, K. R., Mathias, R. A., Nicolae, D. L., Yanek, L. R., Sun, Y. V., Torgerson, D. G., Rafaels, N., Mosley, T., Becker, L. C., Ruczinski, I., Beaty, T. H., Kardia, S. L., Meyers, D. A., Barnes, K. C., Becker, D. M., Freimer, N. B., and Novembre, J. (2011). Recombination rates in admixed individuals identified by ancestry-based inference. *Nat. Genet.*, **43**(9), 847–853.

Yang, Y., Zhang, J., Hoh, J., Matsuda, F., Xu, P., Lathrop, M., and Ott, J. (2003). Efficiency of single-nucleotide polymorphism haplotype estimation from pooled DNA. *Proceedings of the National Academy of Sciences*, **100**(12), 7225–30.

Yuan, X., Miller, D. J., Zhang, J., Herrington, D., and Wang, Y. (2012). An overview of population genetic data simulation. *J. Comput. Biol.*, **19**(1), 42–54.

Zhang, H., Yang, H.-C., and Yang, Y. (2008). PoooL: an efficient method for estimating haplotype frequencies from large DNA pools. *Bioinformatics*, **24**(17), 1942–8.

Zhou, D., Udpa, N., Gersten, M., Visk, D. W., Bashir, A., Xue, J., Frazer, K. a., Posakony, J. W., Subramaniam, S., Bafna, V., and Haddad, G. G. (2011). Experimental selection of hypoxia-

tolerant Drosophila melanogaster. *Proceedings of the National Academy of Sciences of the United States of America*, **108**(6), 2349–54.