Methods and software to enhance statistical analysis in large scale problems in breeding and quantitative genetics

Dissertation

zur Erlangung des Doktorgrades (Dr. rer. nat.)
der Fakultät für Agrarwissenschaften
der Georg-August Universität Göttingen

vorgelegt von

Torsten Pook
geboren am 24.05.1992 in Minden

Göttingen, im Mai 2019

1. Gutacher: Professor Dr. Henner Simianer, Georg-August Universität Göttingen
2. Gutacher: Professor Dr. Timothy Mathes Beissinger, Georg-August Universität Göttingen
3. Gutacher: Professor Dr. Hans-Peter Piepho, Universität Hohenheim

Tag der mündlichen Prüfung: 27. Juni 2019

Erklärung

1. Hiermit erkläre ich, dass diese Arbeit weder in gleicher noch in ähnlicher Form bereits anderen Prüfungsbehörden vorgelegen hat.
   Weiter erkläre ich, dass ich mich an keiner anderen Hochschule um einen Doktorgrad beworben habe.
2. Hiermit erkläre ich eidesstattlich, dass diese Dissertation selbständig und ohne unerlaubte Hilfe angefertigt wurde.

Göttingen, 09.05.2019 Torsten Pook

Acknowledgments

"There are many things in your heart you can never tell to another person."
Greta Garbo

I would like to thank my supervisor **Henner Simianer** for allowing me to have so much flexibility in my work and so much opportunity for traveling to conferences and research stays around the world. Thank you for challenging me and all the advice and guidance along the way.

I would like to thank **Martin Schlather** for being a supervisor who was willing to do a great deal of the annoying parts of the programming work in all our joint projects and convincing me to do a PhD after my Master.

I would like to thank **Tim Beissinger** for becoming my referee on such a short notice.

I would like to thank all the people from the **RTG 1644**, and in particular **Thomas Kneib**, for bringing in an outside perspective to the problems I have been working on.

I would like to thank **Edward Buckler** for letting me join his lab and letting me learn so much from him and his group.

I would like to thank **Ute Döring**, **Dörte Dede** and all the other people helping to take care of so much of the administrative work I seem to attract and despise.

I would like to thank all partner from the **MAZE** project for the discussions during our meetings and providing such a great environment to work in.

I would like to thank **Johannes Geibel**, **Christian Reimer**, **Johannes Heise**, **Johannes Martini** and all the other members and former members of my working

group who had to endure all my quirks and questions about basic knowledge in genetics.

I would like to thank **Jani de Vos**, **Karl Kremling**, **Andrew Whalen** and all the great people I was allowed to meet on the way of creating this work and that made the way so much more enjoyable.

I would like to thank **Nicholas Schreck** for proofreading of this thesis.

I would like to thank **Hanns-Hermann Lagemann** for fueling my interest in mathematics and being a mentor through life.

I would like to thank my parents **Ingrid** and **Heinrich Pook** for all the support and unconditional love throughout my whole life.

I would like to thank all my **friends** and **family**. It often times seems like I would take all your help, advice and friendship for granted but I am very grateful for about all the support you provided me with.

Funding

I gratefully acknowledge all the financial support I received during my time as a PhD student.

I would like to thank the **German Federal Ministry of Education and Research (BMBF)** for the funding of my project (MAZE - "Accessing the genomic and functional diversity of maize to improve quantitative traits"
(Funding ID: 031B0195)).

I would like to thank the **German Research Foundatin (DFG)** for funding my Research Training Group 1644 "Scaling problems in statistics" and thereby a lot of my travelling (grant 922 no. 152112243).

I would like to thank the **European Union's Horizon 2020 Research and Innovation** for funding my work in the project IMAGE (grant agreement n 677353).

Contents

Erklärung i

Acknowledgments iii

Summary 3

Zusammenfassung 5

1 Introduction 7
- 1.1 Rise of dimensionality in genomic data . . . 7
- 1.2 Patterns in genetic data: haplotype blocks . . . 8
- 1.3 Imputation . . . 9
- 1.4 Genomic prediction . . . 11
- 1.5 Design of breeding programs . . . 11

2 HaploBlocker 15
- 2.1 Abstract . . . 16
- 2.2 Introduction . . . 16
- 2.3 Materials and Methods . . . 18
- 2.4 Results and Discussion . . . 31
- 2.5 Supplementary Material . . . 45
  - 2.5.1 Supplementary files . . . 45
  - 2.5.2 Supplementary tables . . . 48
  - 2.5.3 Supplementary figures . . . 50

3 Improving imputation quality in BEAGLE 53
- 3.1 Abstract . . . 54
- 3.2 Introduction . . . 54
- 3.3 Materials and Methods . . . 56
  - 3.3.1 Evaluation of inference and UM imputation quality . . . 58
  - 3.3.2 Evaluation of phasing quality . . . 58
  - 3.3.3 Choice of reference panel in UM imputation . . . 58
  - 3.3.4 Data Availability . . . 59
- 3.4 Results . . . 60
  - 3.4.1 Inference quality . . . 60
  - 3.4.2 Phasing quality . . . 64
  - 3.4.3 UM Imputation quality . . . 66
  - 3.4.4 Comparison of reference genomes . . . 68

3.4.5 Use of a genetic map . . . 69
3.4.6 Quality control using Dosage R-Squared . . . 70
3.4.7 Choice of the reference panel . . . 71
3.5 Discussion and Conclusions . . . 73
3.5.1 Significance of improvement . . . 73
3.5.2 Genetic map and DR2 . . . 74
3.5.3 Reference panel . . . 74
3.5.4 Parameter adaptation . . . 74
3.6 Supplementary Material . . . 77
3.6.1 Supplementary tables . . . 77
3.6.2 Supplementary figures . . . 83

4 MoBPS - Modular Breeding Program Simulator 123
4.1 Abstract . . . 124
4.2 Introduction . . . 124
4.3 Methods . . . 125
4.4 Results and Discussion . . . 127
4.5 Web resources . . . 128

5 Discussion 131
5.1 Influence of imputation quality on haplotyping methods . . . 131
5.2 Genomic prediction using haplotype blocks . . . 132
5.3 Design of breeding programs . . . 135
5.3.1 Introgression of a single QTL in chicken . . . 135
5.3.2 Performing a cock rotation to avoid inbreeding in chicken . . . 136
5.3.3 Gene editing in a cow breeding program . . . 138
5.3.4 Generation of MAGIC population in maize . . . 139
5.3.5 Generation of a base population with LD and a hard sweep . . . 140
5.4 Potential of gene editing in breeding . . . 141
5.4.1 Turning the PAGE - the potential of genome editing in breeding for complex traits revisited . . . 143
5.4.2 Further thoughts . . . 148
5.5 Outlook & Conclusions . . . 149
5.5.1 Outlook for HaploBlocker . . . 149
5.5.2 Outlook for MoBPS . . . 150
5.5.3 Concluding remarks . . . 150

6 References 151

7 List of Figures 169

8 List of Tables 179

A Guidelines to HaploBlocker 183

B Guidelines to MoBPS 201

C Supplementary figures **265**

D Further publications **267**

E Curriculum Vitae **271**

"All models are wrong, but some are useful."
George Edward Pelham Box

Summary

The aim of this thesis is the development of methods and software to enhance the statistical analysis in large scale problems in breeding and quantitative genetics. In **Chapter 1** a brief introduction to the subject of big data is given and the topics relevant for the following chapters are presented.

In **Chapter 2** a new method (HaploBlocker) for the identification of haplotype blocks and libraries is presented that is also implemented in the associated R-package HaploBlocker. In contrast to commonly applied methods for the identifying haplotype blocks, HaploBlocker not only utilizes population-wide measures of linkage disequilibrium (LD), such as the correlation between genetic markers, but also analyzes groups of haplotypes for segments with the same genetic origin (identity-by-descent, IBD). Haplotype blocks are defined as a sequence of genetic markers that has a predefined minimum frequency in the population and only haplotypes with a similar sequence of markers are considered to carry that block. Since the identified blocks are subpopulation specific, much longer haplotype blocks than in conventional methods can be identified. This in turn leads not only to a substantial reduction in the number of variables for later analysis, but also to potentially more informative variables than single nucleotide polymorphisms (SNP). By using HaploBlocker a dataset of 501 doubled haploid lines in a European maize landrace genotyped at 501'124 SNPs was reduced to 2'991 haplotype blocks with an average length of 2'685 SNPs. Despite the lower number of variables, 94% of the genetic diversity of the original dataset can be explained by the block dataset.

Steps of quality control must be performed before genetic data can be analyzed in methods such as HaploBlocker. A central part of any quality control protocol is imputation, which is discussed in **Chapter 3**. The phasing accuracy is of central importance for HaploBlocker and is therefore a special focus in the analysis. In addition, the applicability of commonly applied imputation software for livestock and crop datasets is evaluated, as commonly used tools were originally developed for the use in human genetics. In particular, the software BEAGLE is examined here, as it enables the user to adapt the algorithm to the genetic structure of the dataset by tuning parameter settings. The error rates of imputation were reduced by up to 98.5% by parameter tuning such as the effective population size. In addition, further influencing factors for imputation such as the construction of a suitable reference dataset and the choice and validation of the used reference genome were considered.

In **Chapter 4** the software MoBPS (Modular Breeding Program Simulator) that was developed within the scope of this thesis, is presented. MoBPS is an R-package

that can assist scientists and breeders to simulate both breeding programs and historical populations. Among others, resulting breeding programs can be compared in terms of their economic impact, resulting genetic gain and inbreeding. MoBPS uses a modular and flexible design that allows for the simulation of different breeding programs, but is still very efficient in terms of computing time and memory usage.

In the first part of the discussion (**Chapter 5**) the influence of imputation on the structure of different haplotyping methods is discussed and subsequently the use of HaploBlocker for genomic prediction is analyzed. In the second part of the discussion, different breeding programs that can be simulated via MoBPS are showcased and potential analyses that can be performed based on these simulations are briefly discussed. Particular attention will be paid to the use of genome editing to accelerate the genetic progress for quantitative traits. In the third and last section of this chapter, an outlook on possible further application areas for HaploBlocker and MoBPS is given.

In the supplementary of this thesis, the user manuals for the two R-packages developed in this work are given (**Supplementary A** and **B**).

Zusammenfassung

Das Thema dieser Arbeit ist die Entwicklung von Methoden und Software für die Zucht und die quantitative Genetik um statistische Probleme zu bewältigen, die im Zusammenhang mit immer größer werden Datensätzen und komplexerer Fragestellungen auftreten. In **Kapitel 1** wird eine kurze Einführung in das Thema Big Data gegeben und die für die folgenden Kapitel relevanten Themen werden vorgestellt.

In **Kapitel 2** wird eine neue Methode (HaploBlocker) zur Identifizierung von Haplotypenblöcken und -bibliotheken aufzeigt, die im zugehörigen R-Paket HaploBlocker implementiert ist. Im Gegensatz zu gängigen Methoden zur Identifizierung von Haplotypenblöcken nutzt HaploBlocker nicht nur populationsweite Maße des Kopplungsungleichgewichts (linkage disequilibrium, LD), wie die Korrelation zwischen Markern, sondern analysiert zudem Gruppen von Haplotypen auf Segmente mit gleichem genetischen Ursprung (identity-by-descent, IBD). Ein Haplotypenblock ist definiert als eine Sequenz von genetischen Markern, die mit einer vordefinierten Mindestfrequenz in der Population auftritt und nur Haplotypen mit ähnlicher Sequenz von Markern tragen entsprechenden Block. Da die identifizierten Blöcke in HaploBlocker subpopulationsspezifisch sind, können wesentlich längere Haplotypenblöcke als in herkömmlichen Methoden identifiziert werden. Dies wiederum führt nicht nur zu einer deutlichen Reduzierung der Anzahl der Variablen für die nachfolgende Analysen, sondern auch zu potenziell aussagekräftigeren Variablen als einzelne Marker (single nucleotide polymorphism, SNP). Der Nutzen von HaploBlocker wird durch die Anwendung auf einen Datensatz von 501 doppelhaploider Linien einer Europäischen Maislandrasse mit 501'124 SNPs verdeutlicht. Der entsprechende Datensatz konnte durch Nutzung von HaploBlocker auf 2'991 Haplotypenblöcke mit einer durchschnittlichen Länge von 2'685 SNPs reduziert werden. Trotz der geringeren Variablenzahl können durch den Blockdatensatz noch 94% der genetischen Diversität des Ursprungsdatensatzes erklärt werden.

Bevor genetische Daten mit Methoden wie HaploBlocker analysiert werden können, ist die Durchführung der Qualitätskontrolle erforderlich. In **Kapitel 3** wird mit der Imputation ein zentraler Bestandteil der Qualitätskontrolle genauer beleuchtet. Die Phasinggenauigkeit ist für HaploBlocker von zentraler Bedeutung und ist somit ein besonderer Schwerpunkt in der Analyse. Darüber hinaus wurde zunächst grundsätzlich die Anwendbarkeit von Imputationstechniken für Datensätze aus der Tier- und Pflanzenzucht überprüft, da gängige Methoden für den Einsatz in der Humangenetik entwickelt wurden. Insbesondere die Software BEAGLE wird hier näher betrachtet, da sie es dem Benutzer ermöglicht durch das Anpassen von Inputparametern den Algorithmus auf die genetische Struktur des Datensatzes anzupassen.

Die Fehlerraten der Imputation können durch Parameteranpassungen, wie der effektiven Populationsgroße, um bis zu 98.5% reduziert werden. Darüber hinaus werden weitere Einflussfaktoren für die Imputation, wie die Auswahl eines geeigneten Referenzdatensatzes und Referenzgenoms, betrachtet.

In **Kapitel 4** wird die im Rahmen dieser Arbeit entwickelte Software MoBPS (Modular Breeding Program Simulator) vorgestellt. MoBPS ist ein R-Paket, dass es Wissenschaftlern und Züchtern ermöglicht sowohl Zuchtprogramme als auch historische Populationen zu simulieren. Daraus resultierende Zuchtprogramme können anhand ihrer ökonomischen Auswirkungen, aber auch basierend auf ihrem resultierenden Zuchtfortschritt und dem Inzuchtsniveau verglichen werden. MoBPS nutzt ein modulares und flexibles Design, das die Simulation verschiedenster Zuchtprogramme ermöglicht, aber dennoch sehr effizient in Bezug auf Rechenzeit und Speicherauslastung ist.

Im ersten Teil der Diskussion (**Kapitel 5**) wird der Einfluss der Imputation auf die Struktur verschiedener Haplotypisierungsmethoden diskutiert und anschließend der Einsatz von HaploBlocker für die Zuchtwertschätzung analysiert. In zweiten Teil der Diskussion werden verschiedene Zuchtprogramme, die durch MoBPS simuliert werden können, vorgestellt und potentielle nachfolgende Analysen werden kurz diskutiert. Besonderer Augenmerk wird hier auf die Nutzung von Methoden der Genom-Editierung zur Erhöhung des Zuchtfortschritt für quantitative Merkmale gelegt. Im dritten und letzten Abschnitt dieses Kapitels wird ein Ausblick auf mögliche Anwendungsgebiete und Erweiterungen für HaploBlocker und MoBPS gegeben.

Im Anhang dieser Arbeit werden die User-Manuals für die beiden in dieser Arbeit entwickelten R-Pakete gegeben (**Anhang A** und **B**).

1 Introduction

"The goal is to turn data into information, and information into insight."
Carly Fiorina

In recent years, data collection has become cheaper and easier than ever before. In addition, the available computing resources have increased massively and through this big data analysis has become a field of great importance for modern society. Big data provide opportunities to detect patters that could previously not be identified and thereby could potentially improve the understanding of the underlying truth. However, this also comes with new challenges for scientist to develop methods that are able to cope with the computational and statistical challenges of working with large datasets.

The impact of this rise of big data for genetics are enormous. Scientists today can work with thousands or even millions of genetic markers instead of microsatellites and pedigrees. Rather than focusing on some highly heritable performance traits, a wide range of traits must be taken into account today, with additional challenges caused by correlations between traits, low heritabilities and complex underlying effect structures. In addition, the number of individuals to consider is heavily increasing with national evaluations in cattle using millions of animals. New data types and larger datasets are leading to a paradigm change in the way quantitative genetics is carried out and new tools to perform statistical analysis of these large scale problems are need.

In the following, a general introduction in the structure of genomic datasets will be provided. In addition to that, existing methods in the fields of haplotype blocks, imputation, genomic prediction and the design of breeding programs will be presented and an outlook on the methods and tools developed in this thesis is given.

1.1 Rise of dimensionality in genomic data

In recent years, methods for the generation of data have become more sophisticated and cheaper and by that the size of datasets used in breeding and quantitative genet-

ics has been rapidly increasing. High-throughput phenotyping techniques (Solberg et al., 2006; Cabrera-Bosquet et al., 2012) not only make it possible to phenotype more individuals, but also to consider new traits (Egger-Danner et al., 2015). With regard to the generation of genomic data, the marker density of genotyping arrays has been increasing from low-density (~10K markers, (Boichard et al., 2012)) to medium-density (~50k markers (Matukumalli et al., 2009; Groenen et al., 2009)) to high-density chips (~600k markers, (Matukumalli et al., 2009; Kranis et al., 2013)). Whereas the costs to generate a full genome sequence until 2007 have been more than 10 million US dollars, prices today have dropped to a thousand US dollars (Check Hayden, 2014; Wetterstrand, 2019). Among others, metagenomics (Sleator et al., 2008), epigenomics (Ji et al., 2015), metabolomics (Spratlin et al., 2009) and transcriptomics (Kremling et al., 2018; Herrera-Marcos et al., 2017) are fields of rising popularity with a variety of new data types (omics, (Horgan and Kenny, 2011)) which offer an even wider set of potential variables for analysis.

On the one hand, this growing amount of data sources has the potential to improve prediction and functional understanding of the genome. On the other hand, there are both computational and analytical challenges to overcome (Fan et al., 2014). From a computational point of view, scaling in terms of computational costs and memory requirements needs to be controlled when applying existing and/or developing new methods. Analytical methods need to be robust to handle potential heterogeneity (e.g. joint analysis of multiple subpopulations) that can cause algorithmic instabilities. The interested reader is referred to Fan et al. (2014) for a broader overview of the challenges of big data analysis.

1.2 Patterns in genetic data: haplotype blocks

Patterns in a SNP-dataset can arise from the joint inheritance of physically close segments but can also be caused by functional interactions between genes. The identification and analysis of these patterns is useful for a variety of applications, including the mapping of positive selection (Sabeti et al., 2007; Schrider et al., 2016), the estimation of recombination rates (Nielsen, 2000), and the improvement of the understanding of the underlying genetics of complex traits (Churchill et al., 2004).

A first and intuitive way to detect non-random associations is the use of population-wide measures of linkage disequilibrium (LD). The most commonly used LD measurement r^2 is the analysis of pairwise correlations between markers (VanLiere and Rosenberg, 2008):

$$r^2(p_a, p_b, p_{ab}) = \frac{(p_{ab} - p_a p_b)^2}{p_a\,(1 - p_a)\,p_b\,(1 - p_b)},$$

where p_a and p_b are allele frequencies of alleles a and b at their respective locus and p_{ab} is the frequency of both alleles occurring jointly. However, reducing something as complex as the identification of patterns to a simple correlation can lead to a severe loss of information and thus leads to a multitude of problems (Slatkin, 2008).

As an alternative to identifying patterns at the population level, it is a common practice to screen pairs of haplotypes for sequences of SNPs that are identity-by-descent (IBD, (Donnelly, 1983)). The identification of IBD can be used for imputation (Browning and Browning, 2011; Sargolzaei et al., 2014) and help to assess rates of inbreeding (runs-of-homozygosity, (McQuillan et al., 2008)).

A haplotype block is commonly defined as a set of physically adjacent markers that is used as a joint variable in further analysis (Daly et al., 2001; Sabeti et al., 2002). The boundaries of the haplotype blocks can be set by using of a fixed number of markers per block or a fixed number of different sequences of alleles per block (Meuwissen et al., 2014). Alternatively, LD measures can assist in the identification process of the block boundaries (Gabriel et al., 2002; Taliun et al., 2014; Kim et al., 2017).

The use of haplotype blocks can improve further analysis, as the high volatility and noisiness of pairwise LD (r^2) is heavily reduced (Wall and Pritchard, 2003). This is particularly relevant for applications that make use of dimension reduction (Pattaro et al., 2008) and thereby tackle the $p \gg n$ problem (Fan et al., 2014). Applications can be found in a variety of fields including fine-mapping in association studies (Druet and Georges, 2010; Islam et al., 2016), genomic prediction (Meuwissen et al., 2014; Jiang et al., 2018) and mapping of positive selection in specific regions of the genome (Sabeti et al., 2002, 2007).

In this thesis, the concept of IBD is extended to derive deterministic and subgroup specific haplotype blocks for large scale datasets (Chapter 2), as IBD-based blocks are usually only represented in a probabilistic way (Browning and Browning, 2007) or methods can only be applied to a limited number of individuals due to computational constrains (Moltke et al., 2011). In addition to the method itself, the associated R-package HaploBlocker (R Core Team, 2017; Pook and Schlather, 2019) was developed within the scope of this thesis.

1.3 Imputation

When generating data, it is a common problem that some of the needed information is missing. Imputation is the field in mathematical statistics that is concerned with replacing missing data points with reasonable values.

When generating genetic data there are several factors that can lead to missing data. Since the generation of full genome sequence data is extremely costly, it is common practice to use a genotyping array or low-coverage sequencing instead and then use imputation techniques to fill in missing entries in the dataset by utilizing genomic information of related individuals. Since it is not relevant for later parts of this thesis, details on the molecular processes (LaFramboise, 2009), read alignment (Burrows and Wheeler, 1994; Langmead et al., 2009), SNP-calling (Rabbee and Speed, 2005) and other steps of quality control (Teo, 2008) are neglected here. After these steps, a pair of alleles is called for most markers and imputation is required to fill in the remaining gaps. This is made even more difficult by potential false

calls (Unterseer et al., 2014) and the fact that assessing the gamete from which a base-pair stems from is not easily possible (phasing). All tools discussed in this thesis were developed for diploid genomes. The interested reader is referred to Su et al. (2008) for an imputation algorithm for other ploidy levels.

The imputation of genotype data was first introduced by Li and Stephens (2003). The basic idea of the algorithm is the fitting of a Hidden Markov Model (HMM,(Baum and Petrie, 1966; Rabiner, 1989)) to the sequence of alleles of a haplotype. Over the years, a variety of different tools with a similar basic framework, but improvements to the computational efficiency for larger datasets (Howie et al., 2009), reference panels (Browning et al., 2018) or modifications for improved modeling have been developed. Among others, improvements to the modeling include the use of coalescent trees (Marchini et al., 2007), haplotype clusters (Scheet and Stephens, 2006) and pre-phasing steps (Howie et al., 2012; Scott et al., 2007; Loh et al., 2016). The interested reader is referred to Das et al. (2018); Marchini and Howie (2010) for an detailed review and comparison between commonly used imputation software.

Even though some of the tools mentioned so far account for parent-offspring relationships, the pedigree only has a subordinate role in these algorithms, as the tools were all developed for the use in human genetics, where pedigrees often are limited in size and completeness. Since pedigrees in animal and plant breeding can be much denser (both w.r.t. depth and family sizes), tools to capitalize on this like AlphaImpute (Hickey et al., 2011), AlphaPeel (Whalen et al., 2018) and FImpute (Sargolzaei et al., 2014) have been developed.

Especially in the field of genome-wide association studies (Klein et al., 2005; Yan et al., 2017), but also in the prediction of heritability (Wainschtein et al., 2019) the use of a higher marker density to improve results has shown to be successful. Overall, the analysis of a subset of preselected markers can lead to an ascertainment bias (Albrechtsen et al., 2010; Geibel et al., 2018). Nevertheless, the higher number of markers has to be weighted against a potentially higher share of errors in the data panel and usefulness of imputation needs to be evaluated for the application at hand.

Imputation and especially phasing accuracy are of high importance for the derivation of haplotype blocks in the method developed in this thesis (HaploBlocker, Chapter 2) and therefore is extensively checked in Chapter 3. Among others, the tuning of parameter settings in the software BEAGLE (Browning et al., 2018) to adapt the algorithm to the specific genetic structure of the respective dataset will be discussed. This is of particular interest since the application of HaploBlocker is mainly intended for the use in livestock and crop datasets with lower genetic diversity than human datasets.

1.4 Genomic prediction

One of the most common applications of SNP-datasets is the use for genomic prediction. The genomic value of an individual is estimated based on its genomic and phenotypic data. Since the number of markers is usually high, fitting a traditional ordinary least square model can lead to potential problems in terms of overfitting. Instead, the base model commonly used in genetics is a so-called linear mixed model (Henderson, 1975) that contains an additional random effect u (Fisher, 1918):

$$y = X\beta + Zu + \varepsilon,$$

where X and Z both are matrices containing a set of regressor variables for prediction. The effects β are assumed to be fixed. In the context of a SNP-dataset all entries are all 0, 1, 2. It is common practice to model genomic data as random effects and assume everything else to be fixed. Furthermore, the random effect can be rewritten as the genomic effect $g := Zu$ (Habier et al., 2007). After normalization, the variance of g is given according to the genomic relationship matrix G (VanRaden, 2008). This genomic best linear unbiased predicton (GBLUP) approach was first proposed by Meuwissen et al. (2001).

Today, variations of the model have been successfully implemented in both animal (Hayes et al., 2009; Hayes and Goddard, 2010; Gianola and Rosa, 2015) and plant breeding (Jannink et al., 2010; Albrecht et al., 2011; Nakaya and Isobe, 2012; Heslot et al., 2015). Variations of the model include adaptations to control for inbreeding (Nielsen et al., 2011), accounting for non-additive effects (Da et al., 2014; Jiang and Reif, 2015; Martini et al., 2017), other omics (Li et al., 2019) and haplotype blocks (Meuwissen et al., 2014; Jiang et al., 2018).

Depending on the application, the aim of genomic prediction can be the prediction of phenotypes instead of genetic breeding values. In contrast to prediction of breeding values, which are additive by definition, the accuracy of phenotype prediction can be increased by the inclusion of non-additive effects (Tyler et al., 2016; Forsberg et al., 2017).

In this thesis, the use of haplotype blocks derived in HaploBlocker (Chapter 2) for genomic prediction will be discussed in Chapter 5.2. Furthermore, the use of genomic prediction is of fundamental importance for the design and simulation of breeding programs (Chapter 4).

1.5 Design of breeding programs

Breeding programs are needed to improve the genetic ability of animals and plants w.r.t. productivity, fitness and adaptation. Progress towards the target is limited by the available resources, but also negative effects, such as inbreeding depression or health issues, have to be avoided or at least controlled. Among others, the following components need to be considered (Henryon et al., 2014):

1. Breeding objective
2. Infrastructure
3. Genotyping
4. Phenotyping
5. Prediction
6. Selection
7. Mating
8. Interacting components

The interested reader is referred to Henryon et al. (2014) for details on the individual components and how the use of genomic data can improve them.

In the simplest case, using truncation selection based on phenotypes and a random mating design, the response to selection R per generation can be expressed via the breeders equation (Falconer and Mackay, 1996):

$$R = i \cdot h \cdot \sigma_a.$$

Here i is defined as the selection intensity that is applied, h is the square root of the narrow sense heritability and σ_a is the additive genetic standard deviation. Depending on the application, a variety of extensions to this formula are possible. The use of a breeding value estimation (Chapter 1.4) can lead to a higher precision in the selection of individuals. Depending on the mating scheme a shorter generation interval might be obtainable, leading to a higher gain in the same time frame (Schaeffer, 2006). With rising model complexity, parameters can potentially interact. Most prominently, an increasing selection intensity i will reduce the additive genetic standard deviation σ_a in the following generation (Bulmer-effect, (Falconer and Mackay, 1996)).

Depending on the species, different mating schemes can be applied to achieve the respective breeding objectives. In the simplest case, one can distinguish between purebred and crossbred breeding schemes to account for additive and non-additive genetic effects. For further details the interested reader is referred to Falconer and Mackay (1996); Henryon et al. (2014); Willam and Simianer (2017).

Under several simplifications one can derive closed-form expressions for expected inbreeding levels and genetic progress per cohort of individuals. An example for a software to perform these calculations is ZPLAN+ (Täubert et al., 2010). With rising model complexity and flexible design choices, the derivation of a close-form solution for an optimal allocation of resources becomes virtually impossible. An alternative to this deterministic approach and costly real-world experiments is the use of stochastic simulation. Breeding programs can be simulated in tools like QMSim (Sargolzaei and Schenkel, 2009) and AlphaSim (Faux et al., 2016), followed up by numerical optimization comparison of different breeding schemes for the respective

breeding objective. The use of simulation enables a relatively simple comparison of breeding programs with low costs and potential uncertainty. Nevertheless, a simulation study should always be taken with caution since its is limited to the assumed model, which is usually a simplification of the reality ("All models are wrong, but some are useful", George Edward Pelham Box). Before drawing conclusion from a simulation study it is important to check whether the model assumptions may lead to a bias towards a particular method.

In this thesis, the R-package MoBPS (Modular Breeding Program Simulator, (R Core Team, 2017; Pook et al., 2018)) has been developed (Chapter 4). MoBPS uses a modular and flexible design that allows for the simulation of different breeding programs, but is still very efficient in terms of computation time and memory usage. In Chapter 5 a variety of different breeding programs that can be simulated in MoBPS are showcased and potential applications of these simulations are briefly discussed.

2 HaploBlocker: Creation of subgroup specific haplotype blocks and libraries

"A point of view can be a dangerous luxury when substituted for insight and understanding."
Marshall McLuhan

This chapter contains the manuscript "HaploBlocker: Creation of subgroup specific haplotype blocks and libraries" that has been published in the journal *Genetics* (Pook et al., 2019). For reasons of uniformity in this thesis, the journal style is not used in this chapter.

The manuscript focuses on the methodology of HaploBlocker that is implemented in the associated R-package (R Core Team, 2017; Pook and Schlather, 2019). For a more detailed discussion on potential applications like the detection of selection signatures and genomic prediction we refer to Chapter 5.3.5 and 5.2. For a discussion on potential problems of the algorithm caused by imputation errors it is refer to Chapter 5.1. For the current version of the R-package and an in-depth user manual the interested reader is referred to `https://github.com/tpook92/HaploBlocker` and Supplementary A.

This manuscript is a joined work of Torsten Pook[1,2], Martin Schlather[2,3], Gustavo de los Campos[4], Manfred Mayer[5], Chris-Carolin Schön[5] and Henner Simianer[1,2].

1: Department of Animal Sciences, Animal Breeding and Genetics Group, University of Goettingen, 37075 Goettingen, Germany
2: Center for Integrated Breeding Research, University of Goettingen, 37075 Goettingen, Germany
3: Stochastics and Its Applications Group, University of Mannheim, 68159 Mannheim, Germany
4: Departments of Epidemiology & Biostatistics and Statistics & Probability, Institute for Quantitative Health Science and Engineering, Michigan State University, MI 48824, USA

5: Plant Breeding, TUM School of Life Sciences Weihenstephan, Technical University of Munich, 85354 Freising, Germany

Author contributions by TP

TP lead the development of the methodology, performed most of the analysis, wrote the associated R-package (C parts written by MS), wrote the initial manuscript and led the revision of the manuscript.

2.1 Abstract

The concept of haplotype blocks has been shown to be useful in genetics. Fields of application range from the detection of regions under positive selection to statistical methods that make use of dimension reduction. We propose a novel approach ("HaploBlocker") for defining and inferring haplotype blocks that focuses on linkage instead of the commonly used population-wide measures of linkage disequilibrium. We define a haplotype block as a sequence of genetic markers that has a predefined minimum frequency in the population and only haplotypes with a similar sequence of markers are considered to carry that block, effectively screening a dataset for group-wise identity-by-descent. From these haplotype blocks we construct a haplotype library that represents a large proportion of genetic variability with a limited number of blocks. Our method is implemented in the associated R-package HaploBlocker and provides flexibility to not only optimize the structure of the obtained haplotype library for subsequent analyses, but is also able to handle datasets of different marker density and genetic diversity. By using haplotype blocks instead of single nucleotide polymorphisms (SNP), local epistatic interactions can be naturally modelled and the reduced number of parameters enables a wide variety of new methods for further genomic analyses such as genomic prediction and the detection of selection signatures. We illustrate our methodology with a dataset comprising 501 doubled haploid lines in a European maize landrace genotyped at 501'124 SNPs. With the suggested approach, we identified 2'991 haplotype blocks with an average length of 2'685 SNPs that together represent 94% of the dataset.

2.2 Introduction

Over the years, the concept of haplotype blocks has been shown to be highly useful in the analysis of genomes. Applications can be found in a variety of fields including fine-mapping in association studies (Druet and Georges, 2010; Islam et al., 2016), genomic prediction (Meuwissen et al., 2014; Jiang et al., 2018) and mapping of positive selection in specific regions of the genome (Sabeti et al., 2002, 2007). Haplotype blocks can also be used as a dimension reduction technique (Pattaro et al., 2008; Fan et al., 2014) that produces features which are potentiality more informative

than individual single nucleotide polymorphisms (SNP) (Zhang et al., 2002; Wall and Pritchard, 2003).

Existing methods commonly define a haplotype block as a set of adjacent loci, using either a fixed number of markers or a fixed number of different sequences of alleles per block (Meuwissen et al., 2014). Alternatively, population-wide linkage disequilibrium (LD) measures (Gabriel et al., 2002; Daly et al., 2001; Taliun et al., 2014; Kim et al., 2017) can be used in the identification process to provide more flexibility of the block size based on local genetic diversity. The methods and software (e.g., HaploView, (Barrett et al., 2005)) available for inferring haplotype blocks have become increasingly sophisticated and efficient. Although those approaches to infer haplotype blocks have been proven to be useful, existing methods share some key limitations (Slatkin, 2008). In particular, the use of population-wide measures of LD limits the ability of existing methods to capture cases of high linkage characterized by the presence of long shared segments caused by absence of crossing over (typically within families or close ancestry). To illustrate this, consider the following toy example of four different haplotypes: 11111111, 10101010, 01010101, and 00000000. If all four haplotypes have the same frequency in the population, pairwise LD (r^2) of adjacent SNPs is zero and LD-based algorithms would not retrieve any structure. However, in this example, knowledge of the first two alleles fully determines the sequence in the segment.

In this work, we use the term “allele” for a genetic variant. This can be a SNP or other variable sites like short indels. We use the term “haplotype” for the sequence of alleles of a gamete. This always refers to the full gamete and explicitly not a local and short sequence of alleles. Lastly, a combination of multiple adjacent alleles is here referred to as an "allelic sequence".

As the starting point of our approach (“HaploBlocker”), we assume a set of known haplotypes which can be either statistically determined as accurately phased genotypes, or observed via single gamete genotyping from fully inbred lines or doubled haploids. When the interest is in inferring the longest possible shared segment between haplotypes, a common approach is to identify segments of identity-by-descent (IBD). A tool for the identification of IBD segments is BEAGLE (Browning and Browning, 2013), among others. Since IBD is typically calculated between pairs of individuals, a screening step is used to identify haplotypes that are shared by multiple individuals, e.g. by the tool IBD-Groupon (He, 2013). A method to detect IBD segments directly for groups of individuals has been proposed by Moltke et al. (2011), but is not applicable to datasets with hundreds of haplotypes due to limitations of computing times. A further difficulty is that common methods are not robust against minor variation, leading to actual IBD segments being broken up by calling errors (0.2% with the later used Affymetrix Axiom Maize Genotyping Array (Unterseer et al., 2014)) and other sources of defects.

The imputation algorithm of BEAGLE uses a haplotype library given by a haplotype cluster (Browning and Browning, 2007). The haplotype library in BEAGLE, which is used to initialize a Hidden Markov Model for the imputing step, is only given

in a probabilistic way. This means that there are no directly underlying haplotype blocks that could be used for later statistical application.

Our goal is to provide a conceptualization of haplotype blocks that can capture both population-wide LD and subgroup-specific linkage, and does not suffer from some of the limitations of IBD-based methods. Unlike common definitions that consider haplotype blocks as sets of adjacent markers, we define a haplotype block as an allelic sequence of arbitrary length.

Haplotypes with a similar sequence are locally assigned to the same block. Haplotype blocks are subgroup specific, so that a recombination hot spot appearing in a subgroup of haplotypes does not affect the boundaries of other blocks. This leads to very long blocks, which can cover the same region of the genome, but may vary in the allelic sequence they represent. Even an overlap between the allelic sequences represented by different haplotype blocks is possible.

Subsequently, we start with a large set of identified haplotype blocks and reduce this set to the most relevant blocks and thus generate a condensed representation of the dataset at hand. We define this representation as a haplotype library and, depending on the topic of interest, selection criteria for the relevance of each block can be varied appropriately to identify predominantly longer blocks or focus on segments shared between different subpopulations. The standard input of HaploBlocker is a phased SNP-dataset. In the associated R-package HaploBlocker (R Core Team, 2017; Pook and Schlather, 2019) this input can be provided via the variant call format (VCF, (Danecek et al., 2011)), PLINK Flat files (PED/MAP, (Purcell et al., 2007)) or in a plain matrix object with each column containing a haplotype. For graphical reasons, haplotypes in all examples and figures in the manuscript are displayed in a row. The output of HaploBlocker is a haplotype library that in turn can be used to generate a block dataset. A block dataset contains dummy variables representing the presence/absence of a given block (0 or 1) or, in case of heterozygotes, a quantification of the number of times (0, 1 or 2) a block is present in an individual. The usage of haplotype blocks instead of SNPs allows the use of a variety of new methods for further genomic analyses since the number of parameters is usually massively reduced and haplotype blocks provide a natural model for local epistatic interactions.

2.3 Materials and Methods

The aim of HaploBlocker is to represent genetic variation in a set of haplotypes with a limited number of haplotype blocks as comprehensively as possible. The main idea of our method is to first consider short windows of a given length and increase the length of the analyzed segments in an iterative procedure involving the following steps:

- Cluster-building

- Cluster-merging
- Block-identification
- Block-filtering
- Block-extension
- Target-coverage (optional)
- Extended-block-identification (optional)

Before we elaborate on each step in the following subsections, we give an outline of the three major steps. For a schematic overview of HaploBlocker, we refer to Figure 2.1. In the first step, we derive a graphical representation of the dataset ("window cluster") in which a node represents an allelic sequence and an edge indicates which and how many haplotypes transition from node to node (cluster-building). As locally similar allelic sequences are grouped together, this step also handles robustness against minor deviations (e.g. calling errors). In the second major step, we identify candidates for the haplotype library based on the window cluster. We call this step block-identification and use it to generate a large set of haplotype blocks. In the third and last major step (block-filtering), the set of candidates is reduced to the most relevant haplotype blocks and thereby the haplotype library is generated. In addition to specifying the physical position of each block, we have to derive which haplotypes are included. The fact that blocks are subgroup specific makes the identification of the most relevant blocks complicated so that we split this task into two separate, but closely connected steps (block-identification and block-filtering).

Minor steps in our procedure are cluster-merging and block-extension. The former reduces the computing time in the subsequent steps, whereas the latter increases the precision of the result. However, neither step has a major impact on the resulting haplotype library. Since various parameters are involved in the procedure, their value might be chosen by means of an optimization approach and/or a dataset can be processed with multiple parametrizations in the cluster-building, cluster-merging and block-identification-step. For more details, we refer to the subsections on target-coverage (Supplementary Material File S1) and extended-block-identification.

The next subsections deal with the graphical depiction of the haplotype library and the information loss incurring through the suggested condensation of genomic data. Subsequently, we discuss possible applications, namely the ability of our method to recover founder haplotypes of a population and a block-based version of extended haplotype homozygosity (EHH, (Sabeti et al., 2002)) & integrated extended haplotype homozygosity (IHH, (Voight et al., 2006)). In the last subsection, we introduce the datasets used in this study. Our method, as well as all discussed applications, are available for users by the correspondent R-package HaploBlocker (R Core Team, 2017; Pook and Schlather, 2019). The default settings of the arguments in the R-package correspond to the thread of the following subsections.

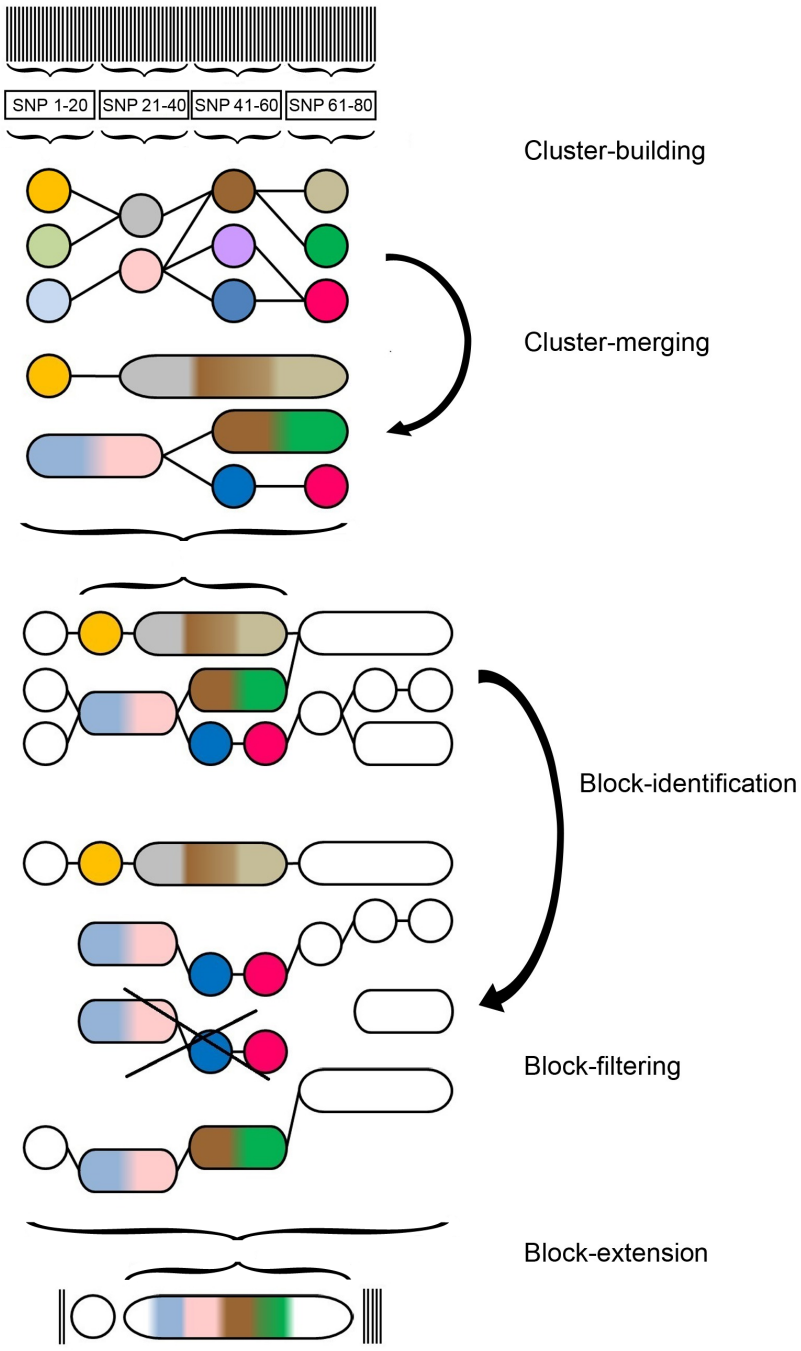


Figure 2.1: Schematic overview of the steps of the HaploBlocker method: (1) Cluster-building: Classifying local allelic sequences in short windows into groups. (2) Cluster-merging: Simplifying window cluster by merging and neglecting nodes. (3) Block-identification: Identifying blocks based on transition probabilities between nodes. (4) Block-filtering: Creating a haplotype library by reducing the set of blocks to the most relevant ones for the later application. (5) Block-extension: Extending blocks by single windows and SNPs. The same allelic sequences in different steps are coded with the same colors in the graph.

Cluster-building

In the first step of HaploBlocker we devide the genome into non-overlapping small windows of size 20 markers as a default value. Accordingly, each haplotype is split into short allelic sequences. To account for minor deviations, we merge groups with similar allelic sequences as follows. For a fixed window, different allelic sequences are considered successively based on their frequency, starting with the most common one. In case a less common allelic sequence differs only in a single marker, they are merged to a group. The allelic sequence of a group ("joint allelic sequence") in each single marker is the most common allele of all contained haplotypes. Usually this will be the most frequent allelic sequence but when allowing for more than one deviation per window this is not necessarily the case anymore. As a toy example, consider a group containing 4x 11111, 3x 10110, 2x 00111 with a resulting joint allelic sequence of 10111. This robustness against errors may lead to actually different haplotypes to be grouped together. In later steps, we will introduce methods to split these haplotypes into different blocks if necessary. The choice of 20 markers as the window size and a deviation of at most one marker as a default is not crucial and should not have a major effect as long as the window size is much smaller than the later identified haplotype blocks. We will present ways to use flexible window sizes in the extended-block-identification-step.

As an example consider a SNP-dataset with 200 haplotypes and 5 markers, given in Table 2.1. The two most common sequences form separate groups (00011 & 11111). For graphical reasons in later steps, we assign 11111 to group 3 even though it is the second group created. The next allelic sequence (11110) is assigned to the group of 11111, as it is only different in a single allele and the joint allelic sequence remains 11111. In case an allelic sequence could join different groups, it is added to the group containing more haplotypes. Based on the groupings we are able to create a graph, called window cluster (Figure 2.2, top graph). Here, each node represents a group (and thus a joint allelic sequence) and the edges indicate how many of the haplotypes of each node transition into which adjacent node.

Table 2.1: Exemplary dataset of allelic sequences and their assignment according to the cluster-building-step.

Frequency	Allelic sequence	Group
101	00011	1
54	11111	3
40	11110	3
3	10011	1
2	01001	2

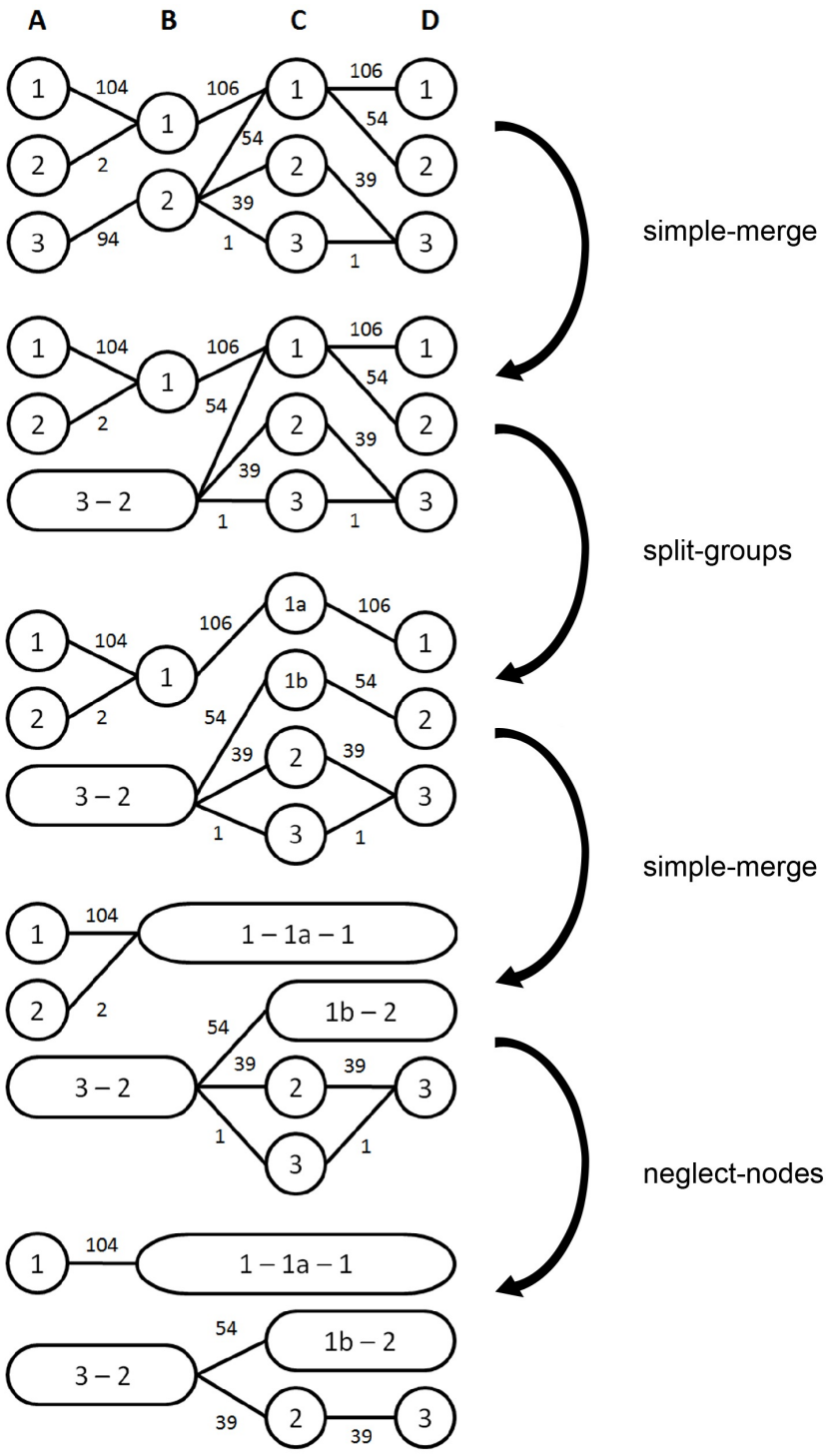


Figure 2.2: The four parts of the cluster-merging-step. Haplotype frequencies in the window A are according to the toy example given in Table 2.1.

Cluster-merging

A window cluster can be simplified without losing any relevant information for later steps of the algorithm. We apply three different techniques:

- simple-merge (SM): Combine two nodes if all haplotypes of the first node transition into the same adjacent node and no other haplotypes are in the destination node.
- split-groups (SG): Split a node into two if haplotypes from different nodes transition into the same node and split into the same groups afterwards.
- neglect-nodes (NN): Remove a node from the cluster if it contains a very small number of haplotypes, 5 say. These removed haplotypes are still considered when calculating transition probabilities between nodes in later steps.

Since the only loss of information in this step stems from neglecting nodes, we first alternately apply SM and SG until no further changes occur. Next, we apply the sequence of NN, SM, SG, SM until fixation of the window cluster. We neglect rare nodes, since a block with few haplotypes (in the most extreme case a block with one haplotype over the whole genome) does not reflect much of the population structure and would have little relevance for further genomic analyses. It should be noted that the minimum number of haplotypes per node in NN does not depend on the number of haplotypes in the sample. This is mainly done to ensure a similar structure of the later obtained haplotype library when adding haplotypes from a different subpopulation. Nevertheless the option to modify this parameter is given, in case one is mostly interested in more common or even rarer allelic sequences.

As an example for the cluster-merging-step consider a dataset with four windows and five different sequences of groups (104x 1111, 54x 3212, 39x 3223, 2x 2111, 1x 3233, Figure 2.2). Haplotypes in the first window are chosen according to Table 2.1. In the first step nodes A3 and B2 are merged by SM. Next, node C1 is split into two nodes via SG. This triggers additional SMs (B1-C1a-D1 and C1b-D2). Afterwards, no SM or SG are possible anymore and NN is performed removing A2 and C3. No further SM or SG are possible after this. Consider that even though D3 is the only node following C2 no SM is possible because removed haplotypes are still considered in later transition probabilities and therefore D3 contains one more haplotype than C2.

Block-identification

In the third step of HaploBlocker we identify the haplotype blocks themselves. As a haplotype block in HaploBlocker is defined as a common allelic sequence in an arbitrarily large window, we use common sequences of nodes in the previously obtained window cluster as a first set of haplotype blocks. The identification process itself is performed by using each node as a starting block. The boundaries of each starting block are given by the boundaries of the node and the allelic sequence is

derived by its joint allelic sequence. A block is iteratively extended if at least 97.5% of the haplotypes in a block transition into the same node; deviating haplotypes are removed. Haplotypes filtered out in this step can rejoin the block if their allelic sequence matches that of the joint allelic sequence of the final haplotype block in at least 99% of the markers. The joint allelic sequence is derived by computing the most common allele in each marker for the contained haplotypes. The choices of 97.5% and 99% worked well in our tests, but any value close but not equal to 100% should work here. This again allows the user some flexibility in how long (in terms of physical length) the haplotype blocks should be and how different jointly considered haplotypes are allowed to be. In a similar way, each edge of the window cluster is used as a starting block. Here, boundaries are given by the boundaries of the two connected nodes. The haplotype blocks identified here will not all be part of the final haplotype library but instead are just a set of candidates from which the most relevant ones will be selected in the block-filtering-step. Note that the share of allowed deviations in this step (1%) is lower than in the cluster-building (1 of 20 markers - 5%), since the size of the identified segment is longer than a single window and the total number of deviations should get closer to the expectation (Unterseer et al., 2014).

To illustrate the method, consider an excerpt of a window cluster given in Figure 2.3. Nodes 2, 3, 4 represent the sequence of groups 3223 of Figure 2.2. When considering the second node as a starting block, we cannot extend the block because there are multiple possible nodes for the contained haplotypes (beforehand: nodes with 88 (93.6%) and 6 (6.4%); afterwards: 54 (57.4%), 1 (1.1%), 39 (41.5%)). When using the fourth node of the excerpt, the block can be extended till the second and fifth node of the cluster since 39 of the 40 haplotypes transition (97.5%) into the same adjacent node. One ends up with the same block when using the third node or the edges including 39, 39 and 40 haplotypes. In case all included haplotypes transition into the same node in the first window, the block could be extended even further. Note that in this step different allelic sequences of the same node (cf. cluster-building-step) can be in different haplotype blocks if they transition into different nodes in later steps (e.g 11111 (54) and 11110 (39+1) in the first window (Table 2.3 & Figure 2.2). For an extension to further increase the size of the set of haplotype blocks, we refer to the extended-block-identification-step.

Block-filtering

After the derivation of a set of candidates in the block-identification, we reduce the set of all haplotype blocks to a haplotype library of the most relevant blocks to represent a high proportion of the dataset with a small number of blocks. First, we compute a rating r_b for each block b that depends on its length (l_b) and the number of haplotypes (n_b) in it:

$$r_b = {l_b}^{w_l} \cdot {n_b}^{w_n}.$$

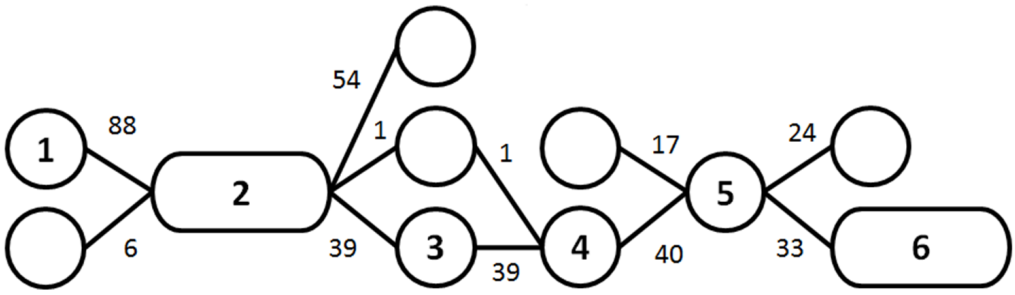


Figure 2.3: Excerpt of a window cluster. This included all edges (transitions) from the nodes of one of the common paths in the example dataset.

Here w_l and w_n represent weighting factors with default values $w_l = 1$ and $w_n = 1$. Note that only the ratio between both parameters matters.

Based on these ratings we determine which haplotype block is the most relevant in each single cell/entry of the SNP-dataset matrix. Iteratively, the blocks with the lowest number of cells as the most relevant block are removed from the set of candidates, until all remaining blocks are the most relevant block in a given number of cells. For this, we will later use the abbreviation MCMB (minimum number of cells as the most relevant block). For our datasets, 5'000 was a suitable value for MCMB but without prior information, we recommend to set a target on what share of the SNP-dataset is represented by at least one block ("coverage"). For details on the fitting procedure we refer to the Supplementary Material (File S1). In case of our example given in Figure 2.3 we end up with block b_1 (green in Figure 2.4) including 94 haplotypes ranging from window 2 to 3 (node 2) with a rating $r_{b_1} = 940$ and block b_2 (red in Figure 2.4) ranging from window 2 to 6 (nodes 2,3,4,5) with a rating $r_{b_2} = 975$. To simplify the example, we assume that no other blocks have been identified. b_2 has a higher rating, therefore cells containing both blocks are counted as cells with b_2 as the most relevant block. This leads to b_1 having 550 cells of the SNP-dataset as the most relevant block and b_2 having 975.

It has to be noted here that the blocks in the final haplotype library can overlap. In case the MCMB is 550 or smaller, overlap occurs in our example and typically can be observed when a short segment is shared in the majority of the population and a smaller subgroup shares a longer segment which includes the short segment. This will lead to dependencies in the presence/absence of blocks that can be addressed in a similar way as linkage disequilibrium between markers.

Even if w_l or w_n is set to zero, there is still an implicit weighting on both the length and the number of haplotypes since each haplotype block has to cover a certain number of cells of the SNP-dataset (MCMB). The overall effect of w_l and w_n is higher when more candidates were created in the block-identification-step.

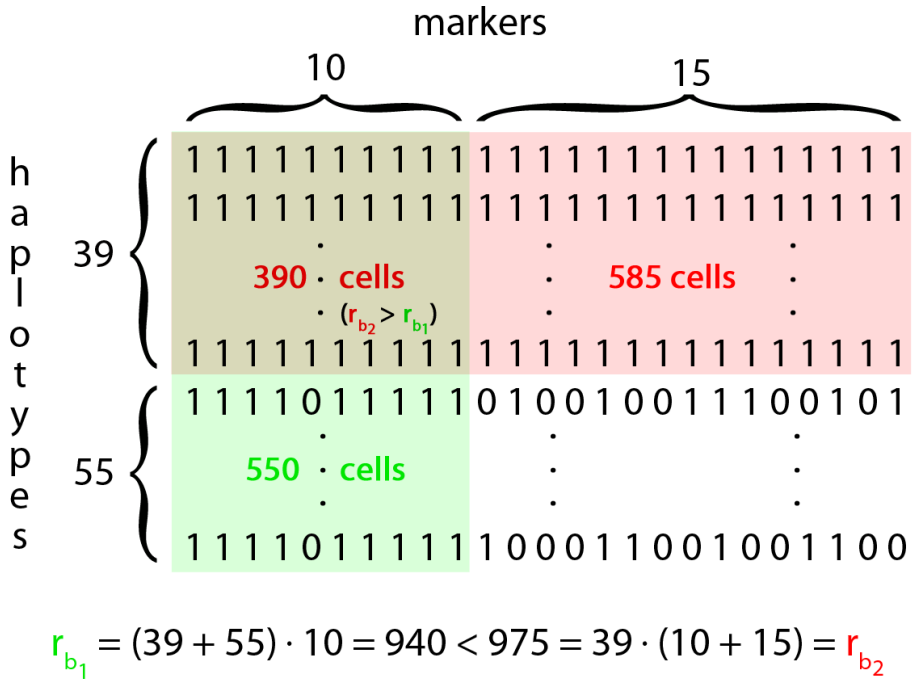


Figure 2.4: Toy example for the calculation in the block-filtering-step with $w_l = w_n = 1$.

Block-extension

The haplotype blocks that have been identified in the previous step are limited to the boundaries of the nodes of the window cluster. Haplotypes in the blocks will transition into different adjacent nodes since the block was previously not extended (cf. block-identification). Nevertheless, different nodes can still have the same allelic sequence in some adjacent windows.

First, haplotype blocks are extended by full windows if all haplotypes are in the same group (cf. cluster-building) in the adjacent window. If the haplotypes of a specific block transition into different groups in the adjacent window, the block is still extended if there is no variation in the following 20 windows. By doing this, we account for possible errors that could have been caused by translocations or phasing issues, for instance. The choice of 20 windows is again rather arbitrary and should be chosen according to the minimum length of the blocks one is interested in. The chosen default results in a relatively large chunk of at least 400 SNPs (20 windows x 20 markers) with all haplotypes of the block required to be classified in the same group for these windows (cf. cluster-building). These conservative settings are chosen because the adjacent segment can also be detected as a separate haplotype block. In any case, all SNPs with variation in a block are identified and reported in the outcome as a possible important information for later analyses.

Secondly, blocks are extended by single adjacent SNPs following similar rules as

the window extension. As a default, we do not allow for any differences here since haplotypes in the block must have some difference in the adjacent window. In case of working with a large number of haplotypes and aiming at identifying the exact end of a block, one might consider allowing for minor differences.

Extended-block-identification (optional)

When extending a haplotype block in the block-identification-step, haplotypes transitioning into a different node are removed. Instead, one could consider both the short segment with all haplotypes and the long segment with fewer haplotypes. As the number of candidates is massively increased, it is recommended to consider the long segment only when at least a share t of haplotypes transition into that node. In our tests $t = 0.95$ was a suitable value for this. Overall, this procedure will lead to the identification of even longer haplotype blocks as candidates for the haplotype library.

To obtain even more candidates in the block-identification-step, one might compute multiple window clusters under different parameter settings (especially concerning window sizes). This provides additional robustness of the method. Especially in case finally obtained haplotype blocks are short, the relevant haplotype blocks can only be identified with a low window size in the cluster-building-step.

Both extensions require substantially more computing time and thus are not included in the default settings of the associated R-package HaploBlocker (R Core Team, 2017; Pook and Schlather, 2019). The R-package contains an adaptive mode using window sizes of 5,10,20,50 markers and a target coverage of 90%.

Graphical representation of haplotype blocks

We suggest a graphical representation of a haplotype library to display transition rates between blocks in analogy to bifurcation plots (Sabeti et al., 2002; Gautier and Vitalis, 2012). This requires ordering of the haplotypes according to their similarity in and around a given marker. For technical details on the sorting procedure we refer to the Supplementary Material (File S2).

Assessment of information content of haplotype blocks

HaploBlocker provides a condensed representation of the genomic data. We next discuss how to quantify the amount of information lost in the process of condensing a SNP-dataset to a block dataset. At a recent conference, de los Campos (2017) proposed three methods for estimating the proportion of variance of an omics set (e.g. high-dimensional gene expression data, methylation or markers) that can be explained by regression on another type of omics data. We used a modified version of the second method proposed by de los Campos (2017) to estimate the proportion

of variance of the full SNP-set genotypes that can be explained by a regression on the blocks of a haplotype library. For the computations in this work the R-packages sommer (R Core Team, 2017; Covarrubias-Pazaran, 2016) and minqa (Powell, 2009) were used with overall very similar results. The methodology can be briefly described as follows:

In traditional SNP-based genomic models (Meuwissen et al., 2001), a phenotype (y) is regressed on a SNP-dataset (X) using a linear model. Entries in X are usually 0,1,2 with dimensionality being the number of individuals (n) times the number of markers (p).

$$y = Xb + \varepsilon,$$

assuming that the markers only have additive effects b. Hence, the vector of genomic values $g = Xb$ is a linear combination of the SNP genotypes. In order to estimate the proportion of g explained by the haplotype library, we regress the genomic values g onto the block dataset represented by a n x q matrix Z, say, of entries 0,1,2. Here q is the number of blocks, with q usually being much smaller than p:

$$g = Za + \delta.$$

From this perspective, genomic prediction based on haplotype blocks searches for a vector Za that is optimal in some sense. For instance, in ridge regression, such a vector is obtained by minimizing a penalized residual sum of squares. It has to be noted that ε is an error term that includes non-genetic effects whereas δ is an error term resulting from genetic effects that cannot be explained by additive effects (a) of single blocks. In random effect models the proportion of the variance of g explained by linear regression on the haplotype library can be estimated using either Bayesian or likelihood methods like REML (Patterson and Thompson, 1971). This proportion of variance explained will vary from trait to trait. We estimate the distribution of the proportion of variance of "genomic vectors" (i.e., linear combinations of SNP genotypes) using a Monte Carlo method. The method proceeds as follows:

1. Sample a vector of weights (b_s) completely at random (e.g. from a standard Gaussian distribution)
2. Compute the underlying genomic value by forming the linear combination: $g_s = Xb_s$
3. Estimate the proportion of variance of g_s that can be explained by regression on haplotype blocks
4. Repeat 1.- 3. for a large number of random vectors b_s

In contrast to commonly used methods like canonical correlation (Witten et al., 2009), this method is asymmetric in that it leads to different results by switching the roles of X and Z. The underlying genomic value is then generated based on the block dataset ($g_s = Zb_s$) and regressed on the SNP-dataset X. Since we compute the share of the variance of one dataset explained by the other dataset, the share

of variation that is not explained can be interpreted as previously underused information. An example for underused information are local epistatic interactions that can be modeled via a block but are usually not fully captured by linear regression on single markers.

Recent work has suggested that the direct estimation of the heritability using REML variance components is biased (Schreck and Schlather, 2018), so we use their proposed estimator. For the traditional estimates using REML estimates as proposed in the conference talk by de los Campos (2017) we refer to the Supplemental Material (Table S1). Overall, results were similar.

Recovering founder haplotypes

HaploBlocker does not require or make use of pedigree or founder haplotypes, but rather provides a method to recover haplotypes from the founders (or a genetic bottleneck) just based on a genetic dataset of the current generation. To evaluate the ability to recover founder haplotypes, we simulated the generation of a multiparent advanced generation intercross population (MAGIC) based on the breeding scheme given in Zheng et al. (2015). Simulations were performed with the R-package MoBPS (R Core Team, 2017; Pook et al., 2018) (available at `https://github.com/tpook92/MoBPS`) with 19 founding haplotypes, intercrossing with a diallel design, four generations of random mating and ten generations of self-fertilization (Zheng et al., 2015). Each generation contains $\frac{19 \cdot 18}{2} = 171$ offspring. Genotypes of founders were assumed to be fully homozygous with uniformly distributed minor allele frequencies and under the assumption of equidistant markers (50k markers, 1 chromosome with a length of 3 Morgan, mutation rate of 10^{-4} in each marker). The haplotype phase of the final generation of offspring was assumed to be known.

Block-based EHH & IHH

Depending on the application, the resulting block dataset can not directly be used, instead slight modifications of the original methodology are needed to adapt computational techniques to the structure of the dataset. An example for this is the extended haplotype homozygosity statistic (EHH, (Sabeti et al., 2002, 2007)). EHH based on SNPs is defined as the probability of a segment between two markers to be in IBD and can be estimated as:

$$EHH = \frac{\sum_i \binom{n_i}{2}}{\binom{N}{2}}.$$

Here N is the total number of haplotypes and n_i is the number of occurrences of a given allelic sequence between the markers. In a second step, IHH (Voight et al., 2006) for a single marker is defined as the integral when calculating EHH of that marker to adjacent markers (until EHH reaches 0.05).

This concept can be extended to an EHH that is based on haplotype blocks (bEHH). Instead of calculating EHH for each marker, segments between the block boundaries $(a_1, a_2, a_3, ...)$ of haplotype blocks are considered jointly. Here a_i is a physical positions (e.g. in base pairs) in the genome. The set of block boundaries contains all start points of blocks, as well as, all markers directly after a block (and not the end point itself). bEHH between segments $[a_i, a_{i+1} - 1]$ and $[a_j, a_{j+1} - 1]$ is then defined as the probability of two randomly sampled haplotypes to belong to the same haplotype block, or at least a block with the same allelic sequence in the window $[a_i, a_{j+1} - 1]$ (with $i \leq j$). bEHH between two markers is set equal to bEHH between the two respective segments. IHH and derived test statistics like XP-EHH or iHs (Sabeti et al., 2007) can then be defined along the same lines as with single marker EHH. For a toy example on the computations necessary to compute EHH and bEHH we refer to the Supplementary Material (File S3 & Figure S1).

Overall, bEHH can be seen as an approximation of EHH. Computing times are massively reduced since bEHH scores only need to be computed between pairs of segments instead of SNPs, overall leading to $\frac{p \cdot (p+1)}{2}$ necessary computations, with p being the number of segments and SNPs, respectively. Secondly, only allelic sequences of different haplotype blocks, instead of individual haplotypes between the two segments need to be compared for each calculation of bEHH.

As minor deviations from the joint allelic sequence of a haplotype block are possible, the usage of bEHH also provides robustness against calling errors and minor deviations.

Genotype data used

We applied HaploBlocker to multiple datasets from different crop, livestock and human populations. In the following, we report results obtained with a dataset of doubled haploid (DH) lines of two European maize (*Zea mays*) landraces ($n = 501$ Kemater Landmais Gelb (KE) & $n = 409$ Petkuser Ferdinand Rot (PE)) genotyped with the 600k Affymetrix® Axiom Maize Genotyping Array (Unterseer et al., 2014) containing 616'201 markers (609'442 SNPs and 6'759 short indels). Markers were filtered for being assigned to the best quality class (PolyHighResolution, (Pirani et al., 2013)) and having a callrate of 90% or higher. Since we do not expect heterozygous genotypes for DH lines, markers showing an excess of heterozygosity might result from unspecific binding at multiple sites of the genome. Thus, markers were also filtered for having less than 5% heterozygous calls. This resulted in a dataset of 501'124 usable markers. The remaining heterozygous calls of the dataset were set to NA and imputed using BEAGLE 4.0 (Browning and Browning, 2016) with modified imputing parameters (buildwindow=50, nsamples=50, phase-its=15).

Secondly, we used a dataset containing $n = 48$ S_0 plants from KE being generated from the same seed batch as the DH-lines. Since S_0 are heterozygous this corresponds to $n = 96$ haplotypes. Genotyping and quality control was performed in the same way as for the DH-lines, without heterozygosity filters. After imputation,

an additional phasing step for the S_0 using BEAGLE 4.1 (niterations=15) was performed. In both steps the DH-lines were used as a reference panel. Only markers overlapping with the DH dataset were included. This resulted in a second dataset containing $n = 96$ S_0 and $n = 501$ DH haplotypes of KE and 487'462 markers.

Additionally, we used datasets from the 1000 Genomes Project phase 3 reference panel (1000 Genomes Project Consortium, 2015) containing 5'008 haplotypes with a total of 88.3 million markers.

Data Availability

The genetic data for maize, the associated R-package, the source code and a detailed documentation of the package is available at `https://github.com/tpook92/HaploBlocker`. Genetic data from the 1000 Genomes Project (1000 Genomes Project Consortium, 2015) is available at `ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/`.

Supplemental files are available at FigShare. File S1 provides an additional method section on the fitting procedure to obtain a certain target-coverage. File S2 contains an additional method section on how to sort haplotypes for the graphical representation of haplotype blocks. File S3 provides an additional method section in which a toy example for the calculation of bEHH is discussed. File S4 includes the R-code used to generate an exemplary MAGIC population for the section on recovering founder haplotypes. Table S1 contains the proportion of variance explained between the full SNP-dataset, a SNP-subset and the block dataset using traditional heritability estimation as in de los Campos (2017). Table S2,S3,S4 contain results obtained in Table 2.3,2.4,2.5 when additionally using a target coverage of 95%. Figure S1 contains the dataset used in File S3. Figure S2 gives a comparison of the block structure in HaploBlocker and a bifurcation plot (Sabeti et al., 2002; Gautier and Vitalis, 2012). Finally, Figure S3 provides a comparison of the block structure for different parameter settings of MCMB.

2.4 Results and Discussion

Here, we will focus on the results obtained for chromosome 1 (80'200 SNPs) of the landrace KE. All evaluations were also performed for all other chromosomes and the second landrace (PE) with similar results.

Using the previously described default settings of HaploBlocker, we identified 477 blocks which represent a coverage of 94.4% of the dataset and have an average length of 2'575 SNPs (median: 1'632 SNPs). For the whole genome, we identified 2'991 blocks representing 94.1% of the dataset with an average/median length of 2'685/1'301 SNPs. A graphical representation of the block structure for the first 20'000 markers of the set is given in Figure 2.5. Haplotypes were sorted according to their similarity at SNP 10'000. Since there is only limited linkage between markers

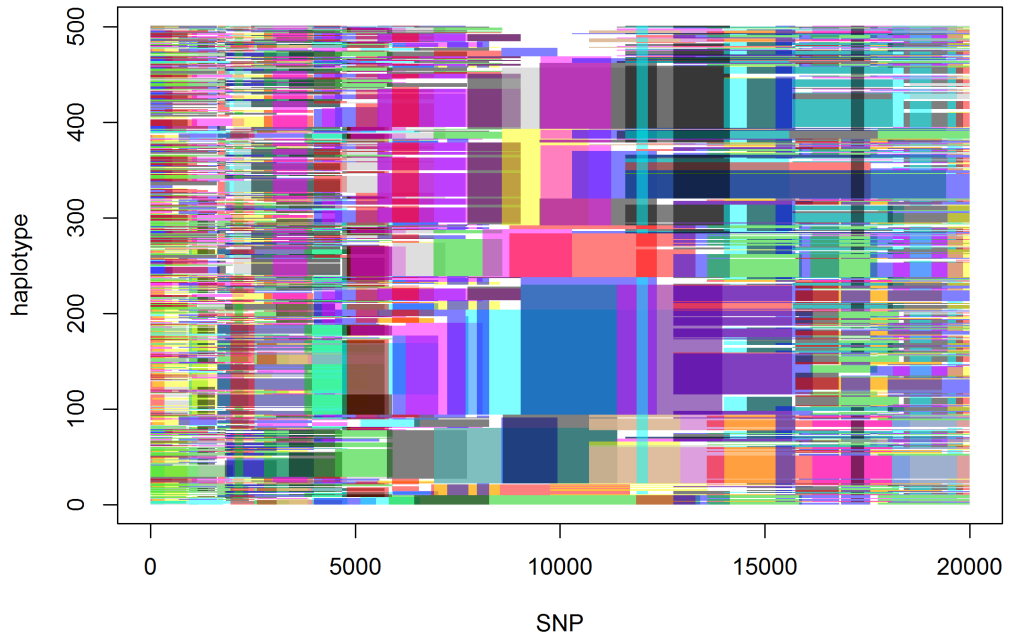


Figure 2.5: Graphical representation of the block structure for the first 20'000 SNPs of chromosome 1 in the KE DH-lines. Haplotypes are sorted for similarity in SNP 10'000. In that region block structures are most visible and transitions between blocks can be tracked easily. Further away from the centre the representation gets fuzzy.

further apart, the graphical representation gets increasingly fuzzy with increasing distance from the target SNP. For a comparison to a bifurcation plot (Sabeti et al., 2002; Gautier and Vitalis, 2012) of that marker, we refer to the Supplementary Material (Figure S2).

When further investigating cells of the SNP-dataset that are not covered by any of the haplotype blocks, one can typically observe that in the associated segments the allelic sequence of the haplotype is a combination of multiple identified haplotype blocks, and by this indicating a recent recombination. Start and end points of blocks can be seen as candidates for positions of ancient (or at least non-recent) recombination, especially when multiple blocks start and end in the same region (e.g between markers 8'572 and 8'601 in Figure 2.5).

In the following, we will show and discuss the influence of certain parameter settings on the resulting haplotype library. Results will be evaluated according to the number of blocks, their length and the coverage of the haplotype library. Note that even

though differences seem quite substantial, most haplotype libraries actually contain the same core set of haplotype blocks, which are the most relevant under basically any parameter setting. Parameter settings mostly influence which of the less relevant blocks are included. By this one can explicitly include a higher share of longer blocks, obtain a certain coverage or similar. For most routine applications, the use of the default settings with a target coverage should be sufficient.

Effect of change in the MCMB

The MCMB affects both the number of blocks and the coverage of the dataset (Table 2.2). Higher MCMB leads to a stronger filtering of the haplotype blocks and thus to a haplotype library with lower coverage and decreased number of larger blocks. Overall, MCMB is the most important parameter to balance between conservation of information (coverage) and dimension reduction (number of blocks). It should be noted that the ideal parametrization of MCMB highly depends on data structure (e.g. marker density). Instead of using a set value for MCMB we recommend to fit the parameter automatically by setting a target coverage. For a graphical comparison of the structure of haplotype libraries with MCMB equal to 1'000, 5'000 and 20'000 we refer to the Supplementary Material (Figure S3).

Table 2.2: Influence of MCMB on the haplotype library for chromosome 1 in the KE DH-lines.

MCMB	Number of Blocks	Average block length (# of SNPs)	Haplotypes per Block	Coverage
1	1'720	1'117	159.9	97.2%
1'000	878	1'892	132.1	96.5%
2'500	621	2'345	120.3	95.6%
5'000	477	2'575	114.9	94.4%
10'000	362	3'022	103.9	92.7%
20'000	274	3'339	99.2	90.1%
50'000	150	3'894	98.5	81.2%

Controlling length and number of haplotypes per block

The window size chosen in the cluster-building-step has a noteworthy influence on the window cluster. By using a smaller window size in the cluster-building-step, the resulting groups are bigger, leading to more and shorter (in terms of physical length) haplotype blocks in the block-identification-step (Table 2.3). As haplotype blocks are much larger than the window size in this case, the effects on the resulting haplotype library are only minor.

Table 2.3: Influence of the window size on the haplotype library for chromosome 1 in the KE DH-lines.

Window size	Number of Blocks	Average block length (# of SNPs)	Haplotypes per Block	Coverage
5	488	2'489	121.8	93.5%
10	482	2'544	113.7	93.6%
20	477	2'575	114.9	94.4%
50	474	2'615	101.4	95.0%

In the block-filtering-step the weighting between segment length (w_l) and number of haplotypes (w_n) in each block influences the structure of the later obtained haplotype library (Table 2.4). As one would expect, a higher weighting for the length of a block leads to longer blocks that include fewer haplotypes. The effect of a lower relative weighting for the number of haplotypes in each block was found to have only a minor effect in our maize data. A possible reason for this is that even when using $w_l = w_n$ the longest blocks previously identified were already selected in the haplotype library.

Table 2.4: Influence of the weighting of block length (w_l) and number of haplotypes (w_n) on the haplotype library for chromosome 1 in the KE DH-lines.

w_l	w_n	Number of Blocks	Average block length (# of SNPs)	Haplotypes per Block	Coverage
1	0	470	2'902	89.3	94.4%
1	0.2	464	2'900	94.1	94.4%
1	0.5	463	2'900	98.4	94.4%
1	1	477	2'575	114.9	94.4%
0.5	1	532	2'218	139.0	94.6%
0.2	1	803	1'518	189.5	95.5%
0	1	1313	934	208.2	96.1%

When using the extended-block-identification method the average length of finally obtained haplotype blocks is massively increased in the obtained haplotype library (Table 2.5). Additionally, overlap between blocks is increased. Using this procedure will lead to the identification of the longest possible IBD segments, making it especially useful for applications like bEHH & IHH.

Evaluations in this subsection were also performed when using a target coverage of 95%. For results on this we refer to the Supplemental Material (Table S2,S3,S4). Overall, results are similar.

Table 2.5: Influence of using the extended-block-identification on the haplotype library in dependency of the parameter t of the extended-block-identification-step for chromosome 1 in the KE DH-lines.

t	Number of Blocks	Average block length (# of SNPs)	Haplotypes per Block	Coverage
1	477	2'575	114.9	94.4%
0.95	603	5'659	89.8	94.6%
0.9	788	9'371	70.2	95.2%
0.8	916	11'716	60.9	95.5%
0.6	970	12'430	58.5	95.7%

Haplotypes out of the sample

To assess how well HaploBlocker identifies haplotype block structures that also pertain to haplotype structures of other datasets, we split the maize data into a training and testing set and compared the share of both datasets represented by a haplotype library based on the training set alone. In all cases the coverage in the test set was below that of the training set, but with higher numbers of haplotypes in the training set the differences gets smaller. In case of 400 haplotypes in the training set and 101 haplotypes in the test set, the difference in coverage is as low as 2.7% (Figure 2.6) indicating that haplotype libraries derived in a sufficiently large dataset can be extended to individuals outside of the sample if they have similar genetic origin. Similar results were obtained when setting a target coverage (90%) for the test set.

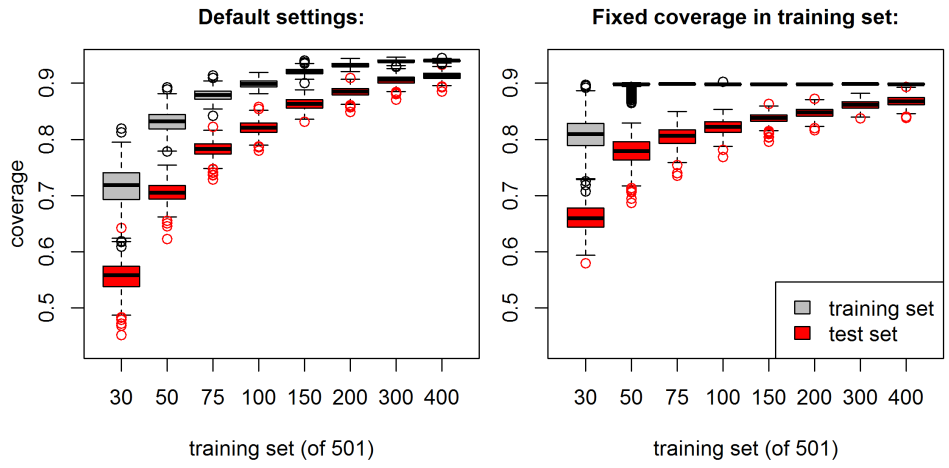


Figure 2.6: Proportion of the dataset represented by the haplotype library (coverage) of the training and test set in regard to size of the training set for chromosome 1 in the KE DH-lines.

Information content

We investigated the information content between SNP- and block-dataset according to the method described above (de los Campos, 2017), where b_s was sampled from a standard Gaussian distribution. A REML approach was used for fitting the model. We found that, on average, 96.0% of the variance of the SNP-dataset can be explained by the default haplotype library (Table 2.6). As one would expect, the share of variance explained is increasing when increasing the number of blocks in the haplotype library. On the other hand, the share of the variance of the haplotype library that can be explained by the SNP-dataset is 95.2%. Even though the number of parameters in the block dataset (Z) is much smaller than in the full SNP set (X), the share of the variance explained by the respective other dataset is similar.

Table 2.6: Proportion of variance explained between the full SNP-dataset (X), a SNP-subset (X_s) and the block dataset (Z). For comparability the number of parameters in X_s and Z were chosen equally.

Number of Blocks/SNPs	$X \sim Z$	$Z \sim X$	$X \sim X_s$
1'720	99.6%	97.8%	99.2%
878	98.6%	96.9%	98.0%
621	97.5%	95.8%	96.8%
477	96.2%	95.3%	95.4%
362	94.8%	94.5%	93.5%
274	92.8%	93.8%	91.0%
150	86.7%	92.0%	82.7%

When using a subset of markers (X_s) with the same number of SNPs as haplotype blocks in the haplotype library, the share of variation explained is slightly lower (95.1%) than for the block dataset. In contrary to the haplotype library, the variation of the SNP-subset is basically fully explained by the full SNP-dataset ($\geq$ 99.99%). This should not be surprising since X_s is a genuine subset of X. Even though a similar share in variation of the SNP-dataset is preserved, the block dataset should be preferred as it is able to incorporate effects that are not explained by linear effects of single markers.

With the following toy example, we illustrate what kind of effects can be grasped by a block dataset compared to a model that is only assigning effects to single markers, as is done in GBLUP (Meuwissen et al., 2001) using the traditional genomic relationship matrix (VanRaden, 2008). Consider a dataset (Table 2.7) with three markers, five haplotypes and a genomic value of 1 for the allelic sequence 111. When assuming no environmental effects, phenotypes equal to genomic values and fitting an ordinary least squares model (OLS) on single markers, the resulting model estimates effects of 0.75, 0.5, and 0.5 for the three respective alleles with an intercept of -1. Overall, single marker effects can approximate but not fully explain an underlying epistatic

genomic value (Table 2.7), whereas a block dataset allows for a natural model of effects caused by local interactions.

Table 2.7: Estimated genomic values using an OLS model assuming additive effects of single markers.

Allelic sequence	Genomic value	Estimated genomic value
111	1	0.75
100	0	-0.25
011	0	0
110	0	0.25
101	0	0.25

Overlapping segments in multiple landraces

When using HaploBlocker on the joint dataset of both landraces (KE & PE), the resulting haplotype library contains essentially the same haplotype blocks that were identified in the haplotype libraries derived for the two landraces individually. The reason for this is that segments shared between landraces are often short, leading to a small rating r_b and thus removal in the block-filtering-step. To specifically identify those sequences that are present in both landraces, we added the constraint that each block had to be present in at least five haplotypes of both landraces. This results in the identification of 1'655 blocks which are present in both landraces. Those blocks are much shorter (avg. length: 207 SNPs) and represent only 62.7% of the genetic diversity of the dataset. This is not too surprising since the haplotypes of a single landrace are expected to be much more similar than haplotypes from different landraces. Explicitly, this is not an indicator for 62.7% of the chromosome of both landraces to be the same. Shared haplotype blocks can be found across the whole chromosome but only some haplotypes of the landraces have those shared segments.

Comparison with the results of HaploView

Overall, the structure of the haplotype blocks generated with our approach is vastly different from blocks obtained with LD-based approaches such as HaploView (Barrett et al., 2005). When applying HaploView on default settings (Gabriel et al., 2002) to chromosome 1 of the maize data, 2'666 blocks are identified (average length: 27.8 SNPs, median: 20 SNPs) and 4'865 SNPs (6.1%) are not contained in any block. If one would use a similar coding to the blocks obtained in HaploBlocker and use a separate variable for each allelic sequence in a block, one would have to account for 12'550 different allelic sequences (excluding singletons). For the whole genome this would result in 16'904 blocks with 79'718 allelic sequences. When using a dataset with both landraces (or in general more diversity), LD-based blocks get even smaller

(for chromosome 1: 4'367 blocks, 24'511 different allelic sequences, average length: 17.3 SNPs, median: 9 SNPs, 4'718 SNPs in no block). In comparison, the haplotype library identified in HaploBlocker with multiple landraces is, with minor exceptions, a combination of the two single landrace haplotype libraries (1'112 blocks, average length: 2'294 SNPs, median: 1'402 SNPs, coverage: 94.4%). Overall, the potential to detect long range associations between markers and to reduce the number of parameters in the dataset is much higher when using haplotype blocks generated by HaploBlocker.

Differences between the two methods become even more drastic when applying HaploView to the human datasets generated in the 1000 Genomes Project Phase 3 (1000 Genomes Project Consortium, 2015). For chromosome 22 there were 49'504 blocks with an average length of 199 SNPs (median: 81 SNPs) that cover 92.9% of the dataset in HaploBlocker. In contrast, there were only 12'304 blocks (excluding singletons) identified in HaploView (average length: 8.1 SNPs, median: 4 SNPs) but only 99'130 of the 424'147 markers were assigned to a block (23.4%). In total, there were still 544'038 different allelic sequences in the identified blocks in HaploView. We noted that all alternative variants were coded as the same allele, as HaploView is only able to handle two alleles per marker, while HaploBlocker is able to handle up to 255 different alleles per marker. When allowing for more than two alleles per marker in HaploBlocker we obtain 49'500 blocks with an average length of 200 SNPs (median: 81 SNPs) that cover 93.0% of the dataset. It should be noted that HaploView was developed with different objectives in mind (Barrett et al., 2005).

Influence of marker density

A common feature of conventional approaches to identify haplotype blocks is that with increasing marker density the physical size of blocks is strongly decreasing (Sun et al., 2004; Kim and Yoo, 2016). To assess this, we executed HaploBlocker on datasets with different marker densities by only including every second/fifth/tenth/fortieth marker of the maize dataset in the model. Since the physical size of a window with a fixed number of markers is vastly different, we compared the structure of the obtained haplotype library using the adaptive mode in HaploBlocker (multiple window clusters with window sizes 5,10,20,50 and adaptive MCMB to obtain a target coverage of 95%) instead of default settings. As there are far fewer markers with possible variation, fewer blocks are needed to obtain the same coverage in the low-density datasets (Table 2.8). Since windows in the cluster-building-step span over a longer part of the genome, the considered groups contain fewer haplotypes leading to less frequent nodes in the window cluster. Since the haplotypes in a node are on average more related to each other, the identified blocks tend to be longer and include fewer haplotypes.

In a second step, we manually adapted the window size (50/25/10/5/5) and the MCMB (5000/2500/1000/500/125) according to the marker density of the dataset. When manually adapting the parameters, the number of blocks in the haplotype

Table 2.8: Structure of the haplotype library under different marker densities using the adaptive mode in HaploBlocker with target coverage of 95% for chromosome 1 in the KE DH-lines.

Density	Number of Blocks	Average block length (# of SNPs on full array)	Haplotypes per Block	Used MCMB
Every SNP	534	2'317	116.4	2'813
Every second SNP	523	2'281	112.7	1'563
Every fifth SNP	450	2'557	96.9	945
Every tenth SNP	401	2'811	90.6	758
Every fortieth SNP	319	3'637	79.9	294

library is largely independent of the marker density (Table 2.9). The length of the blocks is decreasing, whereas the number of haplotypes per block is increasing with decreasing marker density. A possible reason for this is that haplotypes in the same node of the window cluster are less similar in the region than when using bigger window sizes. This will lead to shorter haplotype blocks which are carried by more but less related haplotypes. In case of the dataset in which we used every fortieth marker, we additionally considered a value of 250 for the MCMB since the resulting coverage was a lot higher, indicating that less overall variation is present in the dataset. This also results in fewer overall blocks needed to obtain similar coverage.

Haplotype libraries for all considered marker densities were similar, indicating that for our landrace population a much lower marker density would have been sufficient to derive haplotype blocks via HaploBlocker. In case the physical size of haplotype blocks is smaller, a higher marker density is needed.

Recovering founder haplotypes

HaploBlocker was applied to the final generation of the dataset simulated in analogy to the breeding scheme for the MAGIC population given in (Zheng et al., 2015). On average, we obtained 827 haplotype blocks with a length of 1'420 markers covering 82.8% of the dataset. 96.0% of the allelic sequences of haplotype blocks are at least 99% the same as an allelic sequence of a founder haplotype of that segment. Overall 86.6% of all cells in the SNP-dataset of the founders are recovered by the resulting haplotype library. When using a target coverage of 95%, the share of the allelic sequences of the blocks that are the same as a founder haplotype are quite

Table 2.9: Structure of the haplotype library under different marker densities when adjusting parameters according to data structure for chromosome 1 in the KE DH-lines.

Density	Number of Blocks	Average block length (# of SNPs on full array)	Haplotypes per Block	Coverage
Every SNP	474	2'615	101.4	95.0%
Every second SNP	474	2'720	108.1	95.1%
Every fifth SNP	481	2'557	115.4	95.1%
Every tenth SNP	520	2'174	142.7	95.8%
Every fortieth SNP (MCMB=125)	522	2'056	172.9	97.9%
Every fortieth SNP (MCMB=250)	404	2'287	166.0	96.6%

similar (96.6%) but 93.6% of all cells of the SNP-dataset of the founders can be recovered. Note that identified haplotype blocks, on default, have a minimum size of five haplotypes, leading to the loss of rarely inherited haplotypes.

It should be noted that our approach is not constructed to detect the exact boundaries of IBD segments between founders and single offspring but instead is detecting commonly presented allelic sequences. In a population with limited founders (e.g. caused by a genetic bottleneck), those common allelic sequences most likely stem from the founders of the population. For a plot comparing the true and estimated genetic origin of the final generation we refer to Figure 2.7. Here, estimation means that in case a haplotype block completely stems from a single founder that particular founder is used as the origin. Note that haplotype blocks are much shorter than the size of segments originating from a particular founder, leading to multiple haplotype blocks that all correspond to a part of a segment inherited from a particular founder and therefore same coloring in Figure 2.7. For details on the whole selection procedure we refer to Supplementary File S4. In practice non-overlapping blocks can of course not be assigned to the same founder. Main benefit of our method is that in contrast to commonly used methods only phased genotype data is needed to recover founder haplotypes. When interest is in the exact boundaries of IBD segments for single haplotypes and founders (with known pedigree), we recommend the usage of methods like RABBIT (Zheng et al., 2015).

Figure 2.7: Estimated and true founders for five representative haplotypes of the last generation of a MAGIC population simulated according to breeding scheme given in (Zheng et al., 2015) using a target coverage of 95% in the generation of the haplotype library. Segments are colored according to the originating/estimated haplotype of the founder generation.

Block-based selection signatures

When deriving EHH and bEHH scores, we can observe that curves are quite similar for DH-lines (Figure 2.8). Most apparent difference is a much higher EHH score in the directly surrounding region of the marker. Those segments are typically much smaller than the segments considered jointly in the bEHH approach. Note that the same allelic sequence in such a small region can not only occur based on IBD but also by chance. On the contrary, scores between distant markers for the S_0 plants are much lower when using EHH (Figure 2.8). This is mainly caused by the incorporated robustness of bEHH, since the S_0 dataset tends to contain a higher share of minor deviations between haplotypes.

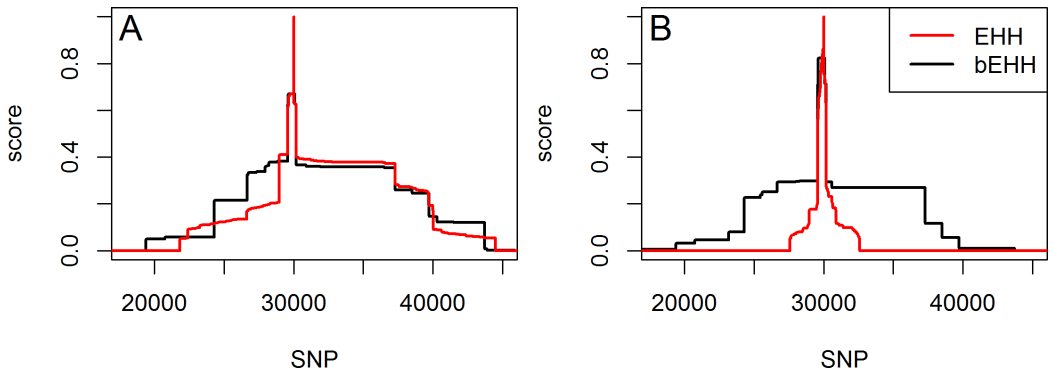


Figure 2.8: Comparision of EHH and bEHH scores for DH-lines (A) and S_0 (B) for marker 30'000 of chromosome 1 in the KE DH-lines.

When using EHH (Sabeti et al., 2002) to derive IHH (Voight et al., 2006), the selection pressure on DH-lines is estimated to be much higher, whereas scores are quite similar between the two groups when using bEHH (Figure 2.9). IHH scores based on bEHH are in concordance with previous research, as we would expect little to no loss of diversity or selection in the process of generating DH-lines (Melchinger et al., 2017). Results in Melchinger et al. (2017) were derived by the use of F_{st} (Holsinger and Weir, 2009) and analysis of molecular diversity in single markers. As presented at a recent conference (Mayer et al., 2018), similar studies with matching results were also performed for KE and PE.

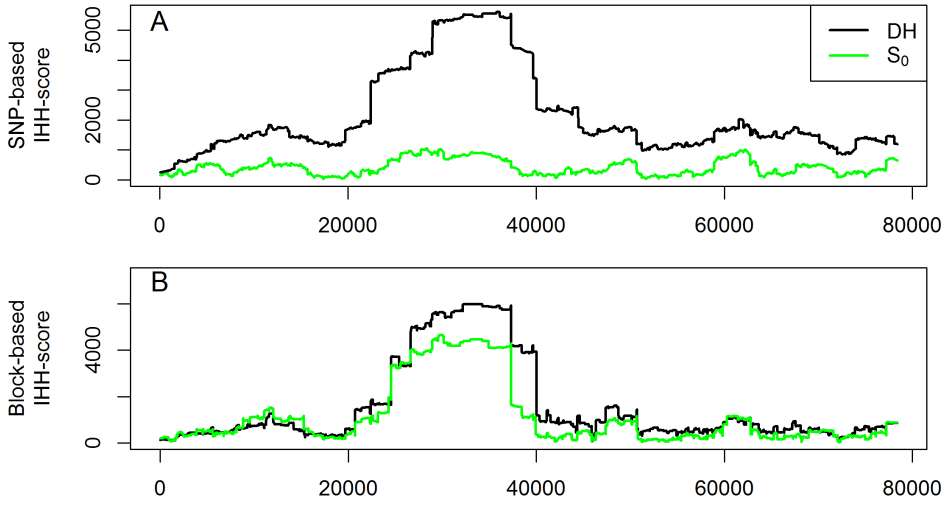


Figure 2.9: IHH scores based on SNPs (A) and haplotype blocks (B) for DH-lines and S_0 for chromosome 1 in the KE.

Computing time

Overall computing times were not an issue for the considered datasets when using the associated R-package HaploBlocker (R Core Team, 2017; Pook and Schlather, 2019) with the full dataset (501 haplotypes, 80'200 SNPs) needing 55 seconds on default, 75 seconds with a target coverage and 13.3 minutes in the adaptive mode. Computations were performed on a single core of a server cluster with Broadwell Intel E5-2650 (2X12 core 2.2 GHz) processors. Most crucial parts in terms of computing time are written in C.

For our datasets, computing time scaled approximately linear in both the number of haplotypes and the physical size of the genome analyzed (Figure 2.10). Especially for the number of haplotypes it is difficult to generalize because the number of nodes in the window cluster is mainly causal for the increase in computing time. The marker

density only had a minor effect. Even a panel containing just every tenth marker, on average, needed 99.3% of the computing time of the full dataset.

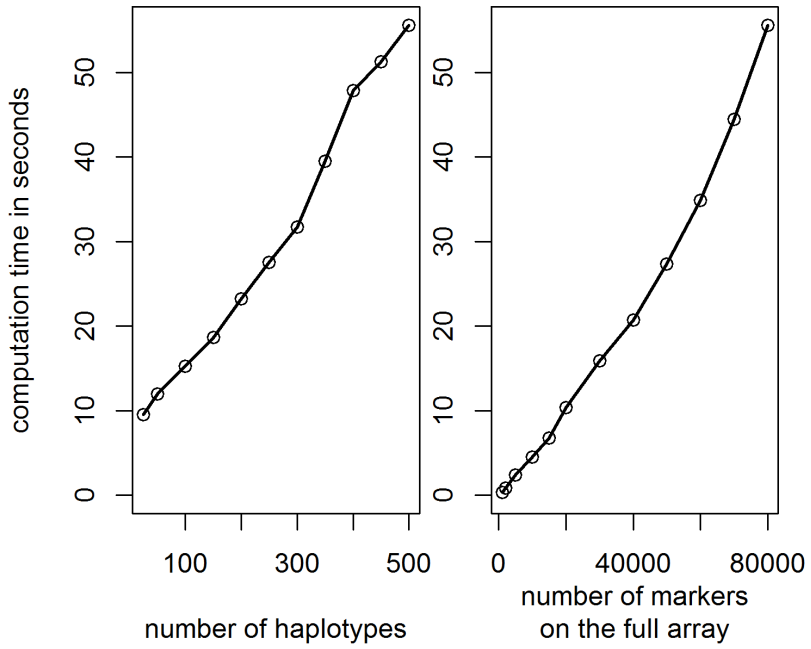


Figure 2.10: Comparison of computing times for datasets of various sizes for chromosome 1 in the KE DH-lines.

Conclusions and Outlook

HaploBlocker provides a natural technique to model local epistasis and thereby solves some of the general problems of markers being correlated but not causal individually (He et al., 2017; Akdemir et al., 2017). This can be seen as one of the factors contributing to the "missing heritability" phenomenon in genetic datasets (Manolio et al., 2009). Obtained haplotype blocks are a concise representation of the variation present in a SNP-dataset. The block assignment in HaploBlocker is deterministic and does not incorporate uncertainty, although the algorithm provides flexibility to control the structure of the haplotype library via parameter tuning.

Even though results were mainly presented for a maize dataset containing DH-lines, methods are not species-dependent nor limited to fully homozygous individuals. Methods were also applied to livestock and human data. As HaploBlocker is not able to handle uncertainty in haplotype phase assignment an initial phasing step is mandatory. For human data in particular, this can be a substantial application problem and therefore requiring triplet data or high quality phase like in the 1000 Genomes Project (1000 Genomes Project Consortium, 2015). Overall, the opportunities for identifying long shared segments will be higher in SNP-datasets

from populations subjected to a recent history of intensive selection as is commonly present in livestock and crop datasets. Recent work has suggested that the phasing accuracy for these kinds of datasets is extremely high (Pook, 2019b) and should therefore be sufficient for the application of HaploBlocker. For datasets containing less related individuals, as commonly present in human data, poor phasing accuracy can limit the applicability and usefulness of HaploBlocker.

It should be noted that by using blocks, an assignment of effects to physical positions (like in a typical GWAS study) is not obtained. A subsequent analysis is needed to identify which segment of the significantly trait-associated haplotype block is causal for a trait and/or which parts of that block differ from the other blocks in that region.

A future topic of research is the explicit inclusion of larger structural variation like duplications, insertions or deletions as is done in methods to generate a pangenome (Eggertsson et al., 2017). Since blocks in HaploBlocker are of large physical size most structural variation should still be modelled implicitly and an application to sequence data is perfectly possible.

HaploBlocker provides an innovative and flexible approach to screen a dataset for block structure. The representation and condensation of a SNP-dataset as a block dataset is enabling new methods for further genomic analyses. For some applications, already existing techniques for a SNP-dataset can directly be applied by using a block dataset instead (e.g. genomic prediction). For other applications, like the detection of selection signatures via EHH/IHH, modifications of the original methodology are needed. Features of HaploBlocker can even enhance existing methods and lead to improvements like an increased robustness of the methods against minor variation or a massively reduced computing time. Additionally, problems regarding typical $p \gg n-$ settings in genetic datasets (Fan et al., 2014) can be heavily reduced, allowing for the usage of more complex statistical models that include epistasis or even apply deep learning methods with a reduced risk of over-fitting.

Acknowledgments

The authors thank the German Federal Ministry of Education and Research (BMBF) for the funding of our project (MAZE - "Accessing the genomic and functional diversity of maize to improve quantitative traits"; Funding ID: 031B0195). Gustavo de los Campos was supported by a Mercator-fellowship of the German Research Foundation (DFG) within the Research Training Group 1644, "Scaling problems in statistics" (grant no. 152112243). We further thank Nicholas Schreck for useful comments on the manuscript and his help regarding unbiased heritability estimation.

2.5 Supplementary Material

The numbering of the Supplementary material corresponds to the order in which they are listed in this section.

2.5.1 Supplementary files

Target-coverage (optional)

In the following, we will denote the share of the dataset that is represented by the haplotype library as the coverage of the dataset. To control the coverage, we propose an adaptive fitting of the MCMB. Especially for different marker densities the choice of the MCMB is relevant to control the minimum size of each block and thus the resulting obtained coverage. The MCMB is fitted by iteratively increasing/decreasing the MCMB when the coverage is too high/low. We double/halve the value of the MCMB from step to step until both a haplotype library with a higher and lower coverage than the target exists. Afterwards the mean of the MCMB values of the two libraries with coverage closes (one above/below) to the target are used next. This procedure is repeated until the MCMB is 1 or the target coverage is reached.

Graphical representation of haplotype blocks

We suggest a graphical representation of haplotype blocks to show transition rates between blocks in analogy to bifurcation plots (Sabeti et al., 2002; Gautier and Vitalis, 2012). To this end, we first sort the blocks of the haplotype library according to the physical position of the first SNP of the block. In case of identical starting points the shorter block is considered first. Our aim in sorting the haplotypes is to cluster haplotypes according to their similarity around a specific physical position (default: SNP in the middle of the dataset). The sorting process itself is executed in two alternating steps:

Step 1: Adding new haplotypes In the first iteration of this step we select all haplotypes in the most common block that includes the marker we want to align against. In later iterations, we add the haplotypes of the block with the biggest overlap of haplotypes with the previously considered block. In case no block has any overlapping haplotypes, the block with the most haplotypes not considered so far is used next.

Step 2: Sorting new haplotypes The newly added haplotypes are ordered according to their presence in physically close blocks. We do this by iteratively comparing the haplotypes of other blocks, starting with the directly adjacent ones. Whenever only some of the currently considered haplotypes are in an adjacent block, we split haplotypes into two groups and proceed with both groups separately. This procedure is stopped when every group has either exactly one haplotype left or the end of the haplotype library has been reached.

Toy example bEHH

As a toy example consider a dataset with 9 haplotypes and 4 haplotype blocks (Supplementary Material Figure S1): green (markers: 1-16), blue (5-16), red (1-20) and purple (11-20). For simplicity we are using marker numbers as physical positions in this example. Resulting block boundaries to consider are 1,5,11,17,21.

EHH between markers 1 and 4 is higher than bEHH for the segment:

$$bEHH([1,4],[1,4]) = \frac{\binom{4}{2}}{\binom{9}{2}} = \frac{1}{6}$$
$$EHH(1,4) = \frac{\binom{4}{2} + \binom{2}{2} + \binom{2}{2}}{\binom{9}{2}} = \frac{2}{9}$$

The higher score is caused by the allelic sequences 0101 and 0110 that both occur twice. Those haplotypes are not part of a haplotype block and thus are ignored in bEHH. For both scores the allelic sequence 0000 and the associated haplotype are present four times.

When considering EHH between markers 5 and 16 or segments [5,10] and [11,16] following scores are obtained:

$$bEHH([5,10],[11,16]) = \frac{\binom{4}{2} + \binom{5}{2}}{\binom{9}{2}} = \frac{4}{9}$$
$$EHH(5,16) = \frac{\binom{2}{2} + \binom{2}{2} + \binom{5}{2}}{\binom{9}{2}} = \frac{1}{3}$$

Allelic sequences 00100111111 and 00000111111 are considered jointly in bEHH, as they are part of the same haplotype block. For EHH, the two allelic sequences are considered separately. Note that this is toy example and in reality blocks are much longer. Haplotypes that are not contained in any block can be considered separated. As this massively increases computing time, this is not done by default.

Script to simulate a diallel design in MoBPS

```
set.seed(7)
nindi <- 19
library(MoBPS) # MoBPS is available at https://github.com/tpook92/
# Alternatively use devtools::install_github("tpook92/MoBPS", subdir="pkg") to install

# Generate a base-population with 50k SNPs, 3 Morgan genome,
# fully homozygous individuals,
# all plants are stored as male individuals (sex=0)
population <- creating.diploid(nindi = nindi, nsnp  =50000, sex.quota = 0,
                               chromosome.length = 3, dataset="homorandom")

# Simulate matings between all founders
population <- breeding.diploid(population, breeding.size = c(nindi*(nindi-1)/2,0),
                               selection.size = c(nindi,0),
                               breeding.all.combination = TRUE,
                               mutation.rate = 10^-4)

# Simulation of 4 generations of random mating of the prior generation
for(index in 1:4){
  population <- breeding.diploid(population, breeding.size = c(nindi*(nindi-1)/2,0),
                                 selection.size = c(nindi*(nindi-1)/2,0),
                                 same.sex.activ = TRUE, same.sex.sex = 0,
                                 mutation.rate = 10^-4)
}

# Simulation of 10 generations of self-fertilization
# Only one offspring per plant
for(index in 1:10){
  population <- breeding.diploid(population, breeding.size = c(nindi*(nindi-1)/2,0),
                                 selection.size = c(nindi*(nindi-1)/2,0),
                                 selfing.mating = TRUE, selfing.sex = 0,
                                 max.offspring = 1, mutation.rate = 10^-4)
}

# Derive haplotypes of last generation and founders
# Founders are double haploid (only one haplotype by plant needed)
haplos <- get.haplo(population, gen = 16)
founderhaplo <- get.haplo(population, gen = 1)[,1:nindi*2]

# Derivation of the haplotype library
library(HaploBlocker)
blockl <- block_calculation(haplos, target_coverage = 0.95)

# Extract points of recombination for final generation in MoBPS:
recombination <- get.recombi(population, gen = 16)
# Compare founder haplotypes to haplotype blocks:
start <- blockl[[1]][[2]]$snp
end <- blockl[[1]][[3]]$snp
concordance <- colMeans(founderhaplo[start:end,]==blockl[[1]][[7]]$snp)
```

2.5.2 Supplementary tables

Table 2.10: Proportion of variance explained between the full SNP-dataset (X), a SNP-subset (X_s) and the block dataset (Z). For comparability the number of parameters in X_s and Z were chosen equally using traditionally heritability estimation as in (de los Campos, 2017).

Number of Blocks/SNPs	$X \sim Z$	$Z \sim X$	$X \sim X_s$
1'720	99.56%	98.14%	99.39%
878	98.59%	97.56%	98.43%
621	97.47%	96.66%	97.51%
477	96.24%	96.20%	96.47%
362	94.79%	95.32%	94.96%
274	92.74%	94.56%	92.94%
150	86.15%	92.68%	86.07%

Table 2.11: Influence of the window size on the haplotype library for chromosome 1 in the KE DH-lines with a target coverage of 95%.

Window size	Number of Blocks	Average block length (# of SNPs)	Haplotypes per Block	Used MCMB
5	682	2'020	138.0	2344
10	646	2'157	125.5	2500
20	537	2'440	119.1	3750
50	474	2'615	101.4	5000

Table 2.12: Influence of the weighting of block length (w_l) and number of haplotypes (w_n) on the haplotype library for chromosome 1 in the KE DH-lines with a target coverage of 95%.

w_l	w_n	Number of Blocks	Average block length (# of SNPs)	Haplotypes per Block	Used MCMB
1	0	530	2'699	96.3	3750
1	0.2	522	2'700	100.3	3750
1	0.5	511	2'694	104.9	3750
1	1	537	2'440	119.1	3750
0.5	1	575	2'136	143.4	4375
0.2	1	722	1'614	186.4	6250
0	1	1030	1'056	206.7	7813

Table 2.13: Influence of using the extended-block-identification on the haplotype library in dependency of the parameter t of the extended-block-identification-step for chromosome 1 in the KE DH-lines with a target coverage of 95%.

t	Number of Blocks	Average block length (# of SNPs)	Haplotypes per Block	Used MCMB
1	537	2'440	119.1	3750
0.95	642	5'484	91.8	4375
0.9	763	9'417	70.1	5313
0.8	848	11'779	62.2	5625
0.6	852	12'576	59.0	6485

2.5.3 Supplementary figures

Figure 2.11: Dataset for the toy example used in File S3

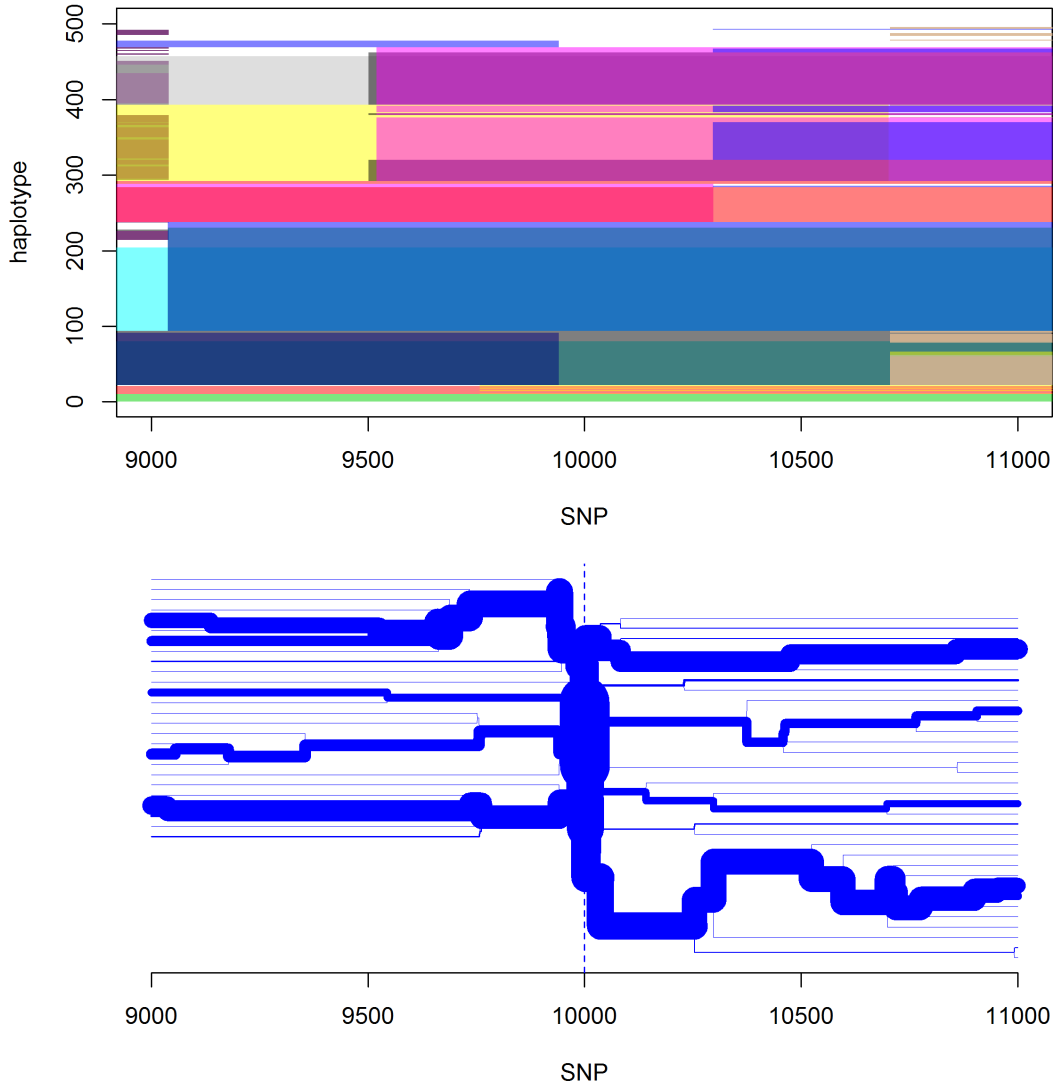


Figure 2.12: Comparison of the block structure and an bifurcation plot (Sabeti et al., 2002; Gautier and Vitalis, 2012) according SNP 10'000 on chromosome 1 in the KE DH-lines.

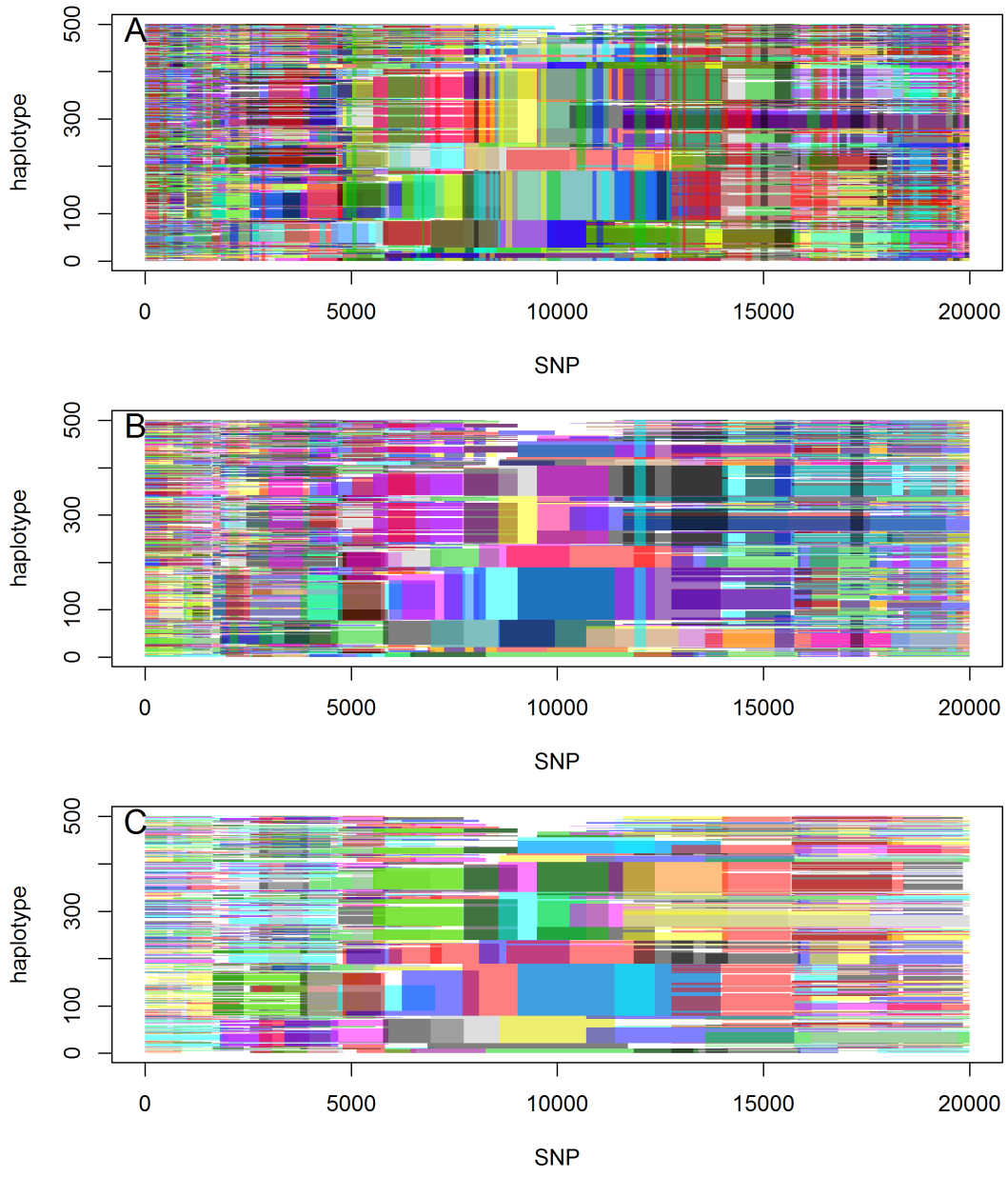


Figure 2.13: Comparision of the block structure for MCMB=1'000 (A), MCMB=5'000 (B), MCMB=20'000 (C) for the first 20'000 SNPs of chromosome 1 in the KE DH-lines.

3 Improving imputation quality in BEAGLE for crop and livestock data

"Data are just summaries of thousands of stories – tell a few of those stories to help make the data meaningful."
Dan Heath

This chapter contains the manuscript "Improving imputation quality in BEAGLE for crop and livestock data" that is currently in review at the journal *G3: Genes, Genome, Genetics*. For reasons of uniformity in this thesis, the journal style is not used in this chapter. For a more detailed discussion on the effect of imputation errors on HaploBlocker (Chapter 2) the interested reader is referred to Chapter 5.1.

This work is a joined work of Torsten Pook[1,2], Manfred Mayer[3], Johannes Geibel[1,2], Steffen Weigend [2,4], David Cavero[5], Chris-Carolin Schön[3] and Henner Simianer[1,2].

1: University of Goettingen, Department of Animal Sciences, Animal Breeding and Genetics Group, 37075 Goettingen, Germany
2: Center for Integrated Breeding Research, University of Goettingen, 37075 Goettingen, Germany
3: Technical University of Munich, Plant Breeding, TUM School of Life Sciences Weihenstephan, 85354 Freising, Germany
4: Friedrich-Loeffler-Institut, Institute of Farm Animal Genetics, 31353 Neustadt-Mariensee, Germany
5: H&N International, 27472 Cuxhaven, Germany

Author contributions by TP

TP lead the development of the methodology, performed most of the analysis, wrote the initial manuscript and led the revision of the manuscript.

3.1 Abstract

Imputation is one of the key steps in the preprocessing and quality control protocol of any genetic study. Most imputation algorithms were originally developed for the use in human genetics and thus are optimized for a high level of genetic diversity. Different versions of BEAGLE were evaluated on genetic datasets of doubled haploids of two European maize landraces, a commercial breeding line and a diversity panel in chicken, respectively, with different levels of genetic diversity and structure which can be taken into account in BEAGLE by parameter tuning. Especially for phasing BEAGLE 5.0 outperformed the newest version (5.1) which in turn also lead to improved imputation. Earlier versions were far more dependent on the adaption of parameters in all our tests. For all versions, the parameter ne (effective population size) had a major effect on the error rate for imputation of ungenotyped markers, reducing error rates by up to 98.5%. Further improvement was obtained by tuning of the parameters affecting the structure of the haplotype cluster that is used to initialize the underlying Hidden Markov Model of BEAGLE. The number of markers with extremely high error rates for the maize datasets were more than halved by the use of a flint reference genome (F7, PE0075 etc.) instead of the commonly used B73. On average, error rates for imputation of ungenotyped markers were reduced by 8.5% by excluding genetically distant individuals from the reference panel for the chicken diversity panel. To optimize imputation accuracy one has to find a balance between representing as much of the genetic diversity as possible while avoiding the introduction of noise by including genetically distant individuals.

3.2 Introduction

Imputation is one of the key steps in preprocessing genetic data generated by SNP-chips or DNA sequencing, as follow-up applications like genomic prediction (Meuwissen et al., 2001) often do not allow for missing values. In some applications the use of a higher marker density can lead to better results even though individuals were not genotyped for most markers (e.g. in genome-wide association studies previously not identified regions can be detected (Yan et al., 2017)).
The imputation of genotype data was first introduced by Li and Stephens (2003). The basic idea of the algorithm is the fitting of a Hidden Markov Model (HMM, (Baum and Petrie, 1966; Rabiner, 1989)) to the sequence of alleles of a haplotype. Over the years, a wide variety of tools with similar basic frameworks, but improvements to the computational efficiency for larger datasets (Howie et al., 2009), reference panels (Browning et al., 2018) or modifications for improved modeling have been developed. Among others, improvements to the modeling include the use of coalescent trees (Marchini et al., 2007), haplotype clusters (Scheet and Stephens, 2006) and pre-phasing steps (Scott et al., 2007; Howie et al., 2012; Loh et al., 2016). To account for the specific structure of livestock and crop datasets, special tools for both cases have been developed. As fully homozygous lines are commonly present in

crops, the software TASSEL (Bradbury et al., 2007) was developed to work well on this data structure (Swarts et al., 2014). Since pedigrees in animal breeding can be much denser than in human populations (both w.r.t. depth and family size), tools like FImpute (Sargolzaei et al., 2014) and AlphaImpute (Hickey et al., 2011) have been developed to fully utilize this information.
In the imputation process all those methods use the fact that physically close markers are likely inherited together, resulting in non-random associations of alleles. These methods thereby rely on the knowledge of the physical position or at least the order of markers for modeling linkage and thus the resulting linkage disequilibrium (LD). In contrast, the software LinkImpute (Money et al., 2015) accounts for LD between pairs of markers and not their physical positions. This can be particularly relevant for species in which no reference sequence is available or whose genomes are known for a high amount of translocations and inversions.
In contrast to other methods using a HMM, the Markov chain in BEAGLE is not initialized by the genotypes or haplotypes themselves, but instead the genetic dataset is used to initialize a haplotype cluster (Browning and Browning, 2007), which subsequently initializes the HMM. In essence, imputation is then performed by identifying the most likely path through the haplotype cluster based on the non-missing genotypes. As BEAGLE was originally developed for application in human genetics, default settings are chosen to work well for imputation in outbred human populations. Nevertheless, the user still has considerable flexibility to tune the algorithm to the specific genetic structure of the respective dataset. As imputation is usually just a step in the preprocessing and quality control protocol, authors tend to use the default settings of a recent version of some imputation software.
To increase the operational marker density via imputation an additional dataset (reference panel) that is genotyped under a higher density can be used. With increasing computational power and more efficient methods available the common advice here is to use as many individuals as possible to get a good representation of the population (Zhang et al., 2013; Browning et al., 2018).
In this paper, we compare different BEAGLE versions (4.0 / 4.1 / 5.0 / 5.1) and perform bench-marking tests in regard to imputation quality on virtually all parameters in BEAGLE for a variety of livestock and crop datasets, as it is one of the most frequently used tools in both animal and plant breeding and a new version of the tool has been recently published (Browning et al., 2018). We further evaluate which individuals to include in a reference panel when aiming at increasing the marker density of a dataset.
Since imputation algorithms like BEAGLE rely on the assumed physical order of markers, the used reference genome influences the imputation quality. Recently, a variety of new maize reference genomes have been made public (Unterseer et al., 2017). We here compare the imputation performance of the commonly used B73v4 (Schnable et al., 2009; Jiao et al., 2017) and new reference genomes from flint lines in maize that should be genetically closer to our material. To this day, all reference genomes derived in chicken were generated based on an inbred Red Jungle Fowl (*Gallus gallus gallus*; (International Chicken Genome Sequencing Consortium, 2004; Bellott et al., 2010).

3.3 Materials and Methods

Genotype data used

In the following, we will consider genotypic data of 910 doubled haploid (DH) lines of two European maize (*Zea mays*) landraces ($n = 501$ Kemater Landmais Gelb (KE) and $n = 409$ Petkuser Ferdinand Rot (PE), (Hölker et al., 2019)) genotyped using the 600k Affymetrix® Axiom® Maize Array (Unterseer et al., 2014). Markers were filtered for being assigned to the highest quality class (Poly High Resolution (Pirani et al., 2013)), having a callrate of at least 90%, and for having at most 5% heterozygous calls, as no heterozygous calls are expected for DH lines. The remaining heterozygous calls were set to NA and subsequently imputed using BEAGLE 4.0 with $nsamples = 50$, resulting in a dataset of 501'124 markers with known haplotype phases.
We further considered two chicken (*Gallus gallus*) datasets genotyped with the 580k SNP Affymetrix® Axiom® Genome-Wide Chicken Genotyping Array (Kranis et al., 2013). Firstly, a chicken diversity panel containing 1'810 chicken of 82 breeds including Asian types, European types, wild types, commercial broilers and layers (Weigend et al., 2014; Malomane et al., 2019). Secondly, a dataset containing 888 chicken of a commercial breeding program from Lohmann Tierzucht GmbH. For quality control SNPs / animals with less than 99% / 95% callrate were removed. We will here focus on chromosome 1, 7 and 20 with 56'773 / 65'177, 12'585 / 13'533 and 5'539 / 5'940 SNPs representing cases for large, medium and small size chromosomes. Remaining missing genotypes for both chicken panels were imputed using BEAGLE 4.1 default.
For tests regarding imputation of ungenotyped markers in maize we used the overlapping markers (45'655 SNPs) of the Illumina® MaizeSNP50 BeadChip chip (Ganal et al., 2011) as a smaller SNP array. As there is no similar public smaller array with a majority of overlapping markers for the chicken panels, we simply used a subset of every tenth marker. All tests regarding imputation quality were performed on imputed datasets. This might favor the respective method used for the imputation. As the missingness in the maize data (1.20%), diversity panel (0.27%) and commercial chicken breeding line (0.32%) were low in the raw data, this effect should only be minor and is neglected here.
To assess the genetic diversity of the three datasets, we derived the LD decay (Figure 3.1) resulting in the highest rates of association for the European maize landraces, followed by the commercial chicken dataset and the chicken diversity panel. The overall genetic diversity in all used datasets should be far smaller than in an outbred human population, which is the data structure BEAGLE was originally developed for. It should be noted that this comparison does not account for possible differences in ascertainment bias (Albrechtsen et al., 2010) between the arrays or the genetic diversity of species and their genomes. Since BEAGLE (and other HMM based imputation methods) are relying on local associations between markers this should still be a good indication for potential imputation performance.

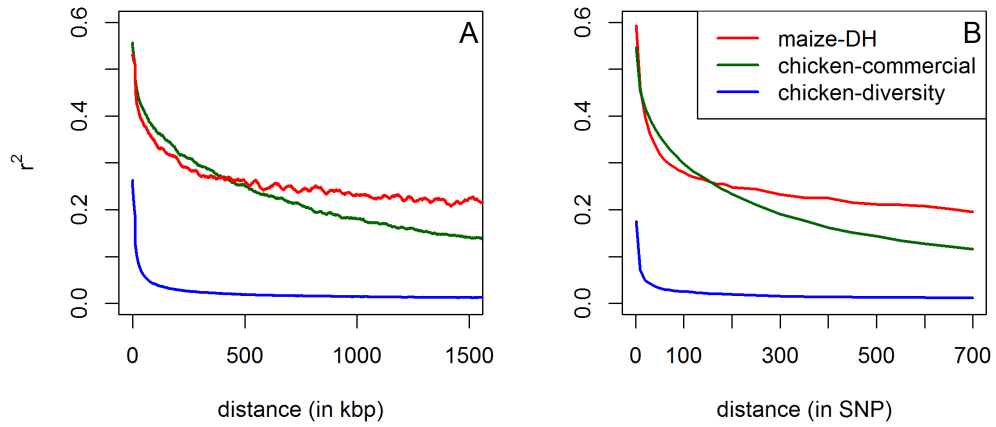


Figure 3.1: LD decay based on physical length (A) and marker distance (B) for chromosome 1 for all considered datasets. Outliers in (A) are corrected for by using a Nadaraya-Watson-estimator (Nadaraya, 1964), using a Gaussian kernel and a bandwidth of 50 kb. (B) is using averaged values for each SNP distance.

Evaluation Pipeline

The imputation process itself can be split up into three internally linked steps which can be of different importance based on the data at hand and, in the following, will be analyzed separately:

1. Inference: All partly or fully missing individual genotypes in the actual dataset are completed, but no additional markers are added.
2. Imputation of ungenotyped markers (UM imputation): Additional markers are added to the genetic data based on information provided by a second dataset (reference panel) with higher marker density.
3. Phasing: The two haplotypes of diploid individuals, i.e. their gametic phases, are estimated from genotype data.

To assess the quality of inference and UM imputation we used the following testing pipeline and repeated the procedure 100 times for each test. We start from a completed dataset in which missing genotypes have been imputed, and consider this as the "true" genotype dataset:

1. Randomly generate missing values (NAs) in the "true" genotype dataset.
   - In case of inference set randomly chosen alleles of all genotypes to NA (in our case: 1% of all alleles with no partly missing genotypes).
   - In case of UM imputation additionally set all entries in a particular marker to NA (maize: according to existing low density array (Ganal et al., 2011); chicken: 90% of all markers).

2. Perform the imputation procedure under a given parameter setting, software and potential use of a reference panel.
3. Evaluation of performance by comparison to the "true" dataset. For more on this we refer to the following subsections.

3.3.1 Evaluation of inference and UM imputation quality

To evaluate the quality of inference and UM imputation we count the total number of entries in the genotype matrix that are different to the "true" dataset (allelic error rate). In this procedure, markers with a low minor allele frequency have a lower impact on the overall quality than in the commonly used practice of calculating the correlation between imputed and "true" dataset (Hickey et al., 2012). To account for this, we will provide error rates depending on the allele frequency as well. A disadvantage of using a correlation is that it does not account for fixed markers as correlation is not defined for those markers, leading to them being excluded from the analysis. As rare variants tend to be more difficult to impute and those variants tend to be fixed at a higher rate, this leads to lower average correlations for methods imputing a rare allele (instead of just imputing the same variant everywhere). Therefore, a fair comparison should only consider those markers that are not fixed over all settings and different software. Especially for UM imputation this would lead to a much smaller set of markers to be considered.

3.3.2 Evaluation of phasing quality

To evaluate phasing quality we use the switch error rate as defined in Lin et al. (2002), which evaluates the number of switches between neighboring heterozygous sites to recover the true haplotype phase compared to the total number of heterozygous markers. Since the true haplotype phase is usually not known the assessment of phasing quality is usually not as straight forward. As we are working with doubled haploid lines in the maize dataset, the true gametic phase is known and a "true" dataset for testing was generated by randomly combining two doubled haploid lines to a Pseudo S_0. The rest of the pipeline can be performed in the same way as for the inference testing. For this analysis, we considered datasets with no missing genotypes to remove any potential noise caused by inference errors.

3.3.3 Choice of reference panel in UM imputation

A common first question when planning to generate genetic data is how many individuals need to be genotyped with high marker density to obtain sufficient imputation quality for individuals genotyped with lower marker density. To evaluate this, we performed imputation on datasets containing 50 individuals as the "true" dataset in our pipeline and generated reference panels containing 25, 50, 100, 150, 200, 250, 300, and 350 individuals, respectively.

Furthermore, we investigate how to chose the individuals to include in a reference panel. This is especially relevant when potential candidates for the reference panel vary in their relationship to the dataset itself. For this, we split the chicken diversity panel into ten subpopulations by iteratively minimizing the total sum of squared genetic distances between breeds within the subpopulations. Distances between the breeds were calculated as Nei standard genetic distances (Nei, 1972). In a first step, the custom made algorithm randomly assigned the breeds to ten equal sized subpopulations. The contribution of each breed to the sum of squared distances was calculated and the algorithm started iteratively exchanging the most noisy breeds to other subpopulations. If there was a reduction of the total sum of squared distances within the subpopulations, the exchange was accepted and the contributions were calculated again. The process was repeated until no exchange could improve the fit. To overcome results depending on specific starting positions, the process was repeated for 60 random starting points. Nei standard genetic distances for evaluation of UM imputation quality of BEAGLE were calculated based on the subpopulation assignment of individuals and UM imputation was performed using the following reference panels:

1. All other individuals of the same subpopulation
2. All individuals of one other subpopulation
3. All individuals of all other subpopulations
4. All individuals of subpopulations with below-average Nei standard genetic distance to the dataset
5. All individuals of those subpopulations with reduced error rates when testing A + B compared to A as the reference panel

Additionally combinations of panels A + B, A + C, A + D and A + E were tested. Tests were repeated 20 times for each subpopulation with datasets containing 50 randomly sampled individuals. For each dataset, all different reference panels were tested. The interested reader is referred to Supplementary Table 3.7 for a detailed list of the used subpopulation assignments and Supplementary Figure 3.11 for the resulting neighbor-joining-tree.

3.3.4 Data Availability

Genetic data for chromosome 1 for all three panels used are available at `https://github.com/tpook92/HaploBlocker`. Table S1 and S2 contains error rates of UM imputation for the commercial breeding line and the diversity panel in chicken. Table S3 provides phasing error rates for the set of Pseudo S_0 with no missing data. Table S4 contains inference error rates for the PE DH-lines using different reference genomes. Table S5 and S6 contain lists of "critical" markers for KE and PE. Table S7 gives error rates of UM imputation using different reference panels for the subpopulations. Table S8 contains the subpopulation assignments for all chicken from

the diversity panel. Table S9 contains the minimal error rates and used parameter settings for all performed tests.
Figure S1 provides the neighbor-joining-tree for the ten subpopulations of the chicken diversity panel. Figure S2 displays the relation between local LD and error rate for chromosome 9 in maize. Figure S3 displays the change in the number of errors in each marker by using low values of *buildwindow*. Figure S4 and S5 display the relation between DR2 and the number of errors per marker. Figure 3.16 - 3.34 display the relation between input parameters and error rates for inference in the maize data. Figure 3.35 - 3.55 display the relation between input parameters and error rates for inference and phasing for the set of Pseudo S_0. Figure 3.56 - 3.84 display the relation between input parameters and error rates for UM imputation for the maize data, the commercial chicken line and the chicken diversity panel.
Supplemental files are available at FigShare: `https://gsajournals.figshare.com/s/4c6ceaa7fe834f0700a3`.

3.4 Results

In the following, obtained error rates of the imputation under a variety of tuning option in BEAGLE are discussed. Here, we consider virtually all available parameters in BEAGLE, the size of the reference panel, and the underlying genetic map. The effect on the error rate of different tuning options are somewhat independent from each other as they commonly affect different parts of the imputation algorithm. Therefore, we will first consider each tuning option individually and later discuss suggested imputation pipelines for the different use cases.
Unless otherwise mentioned, we will report for maize the error rates in the landrace KE averaged over all chromosomes. Results for PE were similar with, on average, slightly increased error rates.

3.4.1 Inference quality

On default, BEAGLE 5.0 (error rate: 0.0142%) and BEAGLE 5.1 (0.0127%) both clearly outperform BEAGLE 4.1 (0.255%) and BEAGLE 4.0 (0.201%) for the maize data. For all four versions the error rates are significantly higher for alleles with low frequency (Figure 3.2). In regard to the location of inference errors one can observe a high volatility with a tendency to have increased error rates in telomeric regions (Figure 3.3). Additionally, error rates in regions of high LD tend to be lower (Supplementary Figure 3.12).

For all four versions the biggest improvement was obtained by tuning parameters that are affecting the structure of the haplotype cluster. The optimal parameter values (Table 3.1) for *buildwindow* (4.0), *singlescale* (4.0), *modelscale* (4.1) lead to less similar haplotypes being clustered jointly. *Phase-segment* (5.0), *phase-states* (5.0 / 5.1) affect the minimum length and number of different haplotypes in the haplotype cluster. Overall, all these settings lead to longer and/or less related haplotypes to

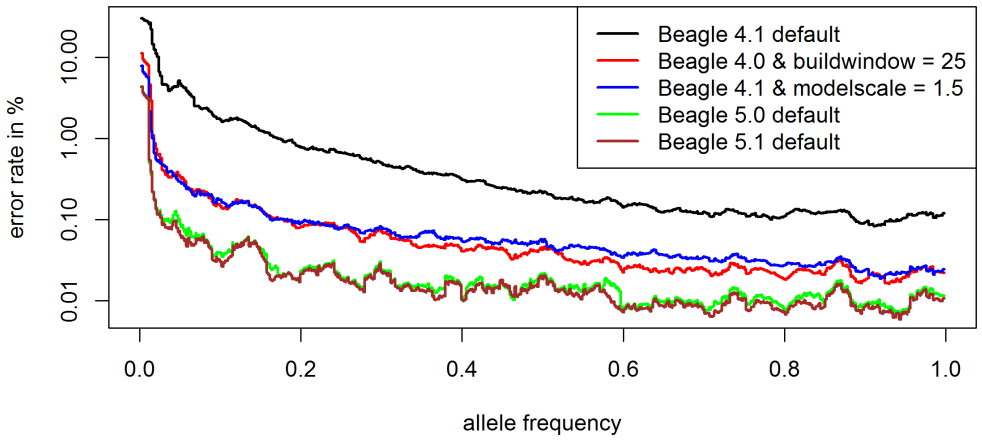


Figure 3.2: Allele specific error rate depending on the allele frequency under different BEAGLE settings for the maize data. Only those dataset entries with the respective allele in the "true" dataset are considered when deriving the allele specific error rate. Y-axis is log-scaled.

be considered jointly. The gains by fitting those parameters are much higher in BEAGLE 4.0 and 4.1 but overall error rates are still higher than in BEAGLE 5.0 and 5.1 (Table 3.1) with BEAGLE 5.1 performing best. Improvements in overall inference quality can be observed for all allele frequency classes and regions in the genome (Figures 3.2 & 3.3). It should be noted that in contrast to later tests in UM imputation the use of low (and probably more realistic) values for *ne* (effective population size) can lead to substantially increased error rates (Figure 3.4). The interested reader is referred to Supplementary Figures 3.16 - 3.34 for the effect on the inference error rate for different parameters. For the maize data the inference error rates were basically unaffected by the number of iterations performed in any of the imputation steps in BEAGLE (Table 3.1). Since the haplotype phase in DH-lines is known and the main purpose of further iterations in BEAGLE is to improve that haplotype phase, this should not be that surprising. After parameter tuning error rates are still lowest in BEAGLE 5.1 with 0.0122% but differences are considerably reduced (BEAGLE 4.0: 0.0307%, BEAGLE 4.1: 0.0436%, BEAGLE 5.0: 0.0132%, Supplementary Table 3.13). Tuning of both *singlescale* and *buildwindow* in BEAGLE 4.0 jointly did not further improve performance with *buildwindow* overall performing better for inference. Even though error rates for extremely low values for *buildwindow* are lowest, this change is not recommended as some markers do show massively increased error rates (Supplementary Figure 3.13).

The inference error rates for the chicken diversity panel are much higher for all versions (~1%) and the relative improvement obtained by adapting parameter settings is lower. As the chicken diversity panel contains more variation and is structurally more similar to outbred human data than the European landraces in maize, this should not be that surprising. With the exception of the parameter *err* the change

Table 3.1: Inference error for the KE DH-lines by changing a single imputing parameter.

Parameter	default	range tested	best	overall impact
BEAGLE 5.1	-	-	default (0.0127%)	-
ne	1'000'000	1 - 1'000'000	100'000 (0.0125%)	Figure 3.17
err	0.000067	0.01 - 0.00001	0.001 (0.0125%)	Figure 3.19
window	40	10 - 1'000	100 (0.0125%)	Figure 3.21
burnin	6	2 - 50	50 (0.0126%)	Figure 3.23
iterations	12	2 - 40	40 (0.0127%)	Figure 3.25
phase-states	280	50 - 10'000	default	Figure 3.28
imp-states, imp-segment, cluster, impstep, impnsteps	1'600, 6, 0.005, 0.1, 7	-	-	only impacts UM imputation
BEAGLE 5.0	-	-	default (0.0142%)	-
ne	1'000'000	1 - 1'000'000	30'000 (0.132%)	Figure 3.16
err	0.0001	0.01 - 0.00001	0.005 (0.0141%)	Figure 3.18
window	40	10 - 1'000	200 (0.0140%)	Figure 3.20
burnin	6	2 - 50	default	Figure 3.22
iterations	12	2 - 50	25 (0.0141%)	Figure 3.24
phase-segment	4	1 - 25	10 (0.0135%)	Figure 3.26
phase-states	280	50 - 1'000	100 (0.0136%)	Figure 3.27
imp-states, imp-segment, cluster, impstep	1'600, 6, 0.005, 0.1	-	-	only impacts UM imputation
BEAGLE 4.1	-	-	default (0.255%)	-
niterations	5	0 - 25	-	virtually no differences for DHs
modelscale	0.8	0.5 - 5	1.5 (0.0438%)	Figure 3.29
ne	1'000'000	1 - 1'000'000	10'000 (0.254%)	Figure 3.30
BEAGLE 4.0	-	-	default (0.201%)	-
buildwindow	1'200	1 - 2'500	5 (0.028%)	Figure 3.31
singlescale	0.8	0.5 - 5	1.5 (0.066%)	Figure 3.32
nsamples	4	1 - 50	50 (0.152%)	Figure 3.33
burnin-its	5	2 - 25	25 (0.199%)	Figure 3.34
phase-its	5	2 - 25	-	virtually no differences for DHs
impute-its	6	2 - 25	-	only impacts UM imputation

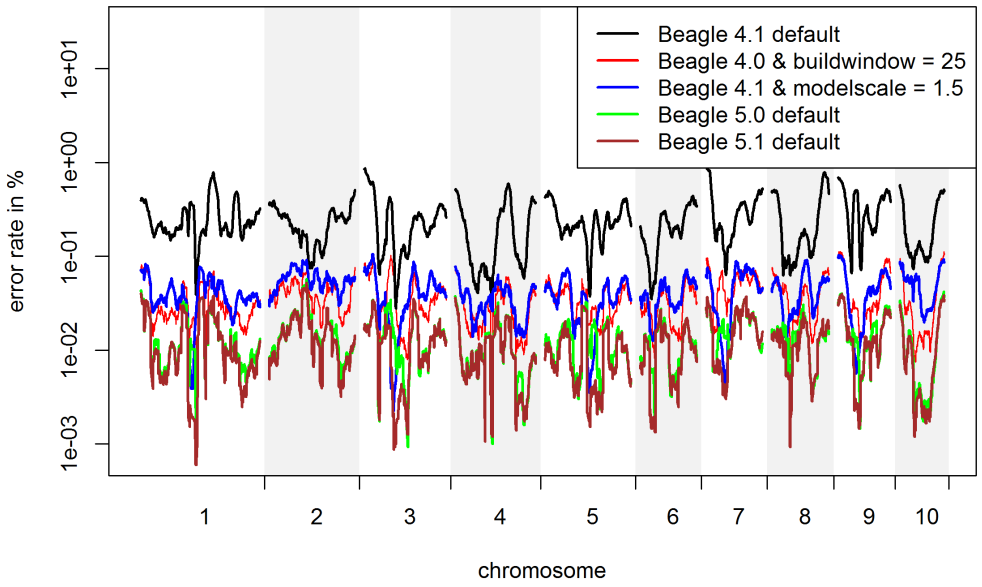


Figure 3.3: Inference error rate based on the location of the genome. Outliers are corrected for by using a Nadaraya-Watson-estimator (Nadaraya, 1964), using a Gaussian kernel and a bandwidth of 3'000 markers for the maize data. Y-axis is log-scaled.

from the default was always in the same direction as for the maize data. As *err* is controlling the allele mismatch probability of known alleles when identifying the most likely path through the haplotype cluster (Browning et al., 2018) this can be seen as an indicator for a higher overall data quality for the maize data. Lowest obtained error rates are 1.01% for BEAGLE 4, 0.80% for BEAGLE 4.1, 0.81% for BEAGLE 5.0 and 0.82% for BEAGLE 5.1 (Supplementary Table 3.13).
Inference error rates for the datasets from the commercial chicken breeding program are between 0.20% and 0.23% for basically all tested settings, leading us to conclude that for inference on this dataset there is not much potential to decrease error rates. A potential reason for this is that other error sources like SNP calling errors may be higher than inference error rates.
When working with the Pseudo S_0 in maize instead, ideal parameter settings are very similar with the key difference of additional gains by increasing the number of iterations performed (Table 3.2). As the algorithm starts with randomly phased genotypes and improves the phase in each iteration, this should again not be surprising. However, excessive *burnin* iterations prior to the actual algorithm only worsened results. The interested reader is referred to Supplementary Figure 3.35 - 3.55 for parameter influences on both inference and phasing quality for the Pseudo S_0. Inference accuracies after parameter tuning are again similar with BEAGLE 5.0 performing best (BEAGLE 4.0: 0.0193%, BEAGLE 4.1: 0.0168%, BEAGLE 5.0: 0.0109%, BEAGLE 5.1: 0.0148%). Note that error rates given in Table 3.2

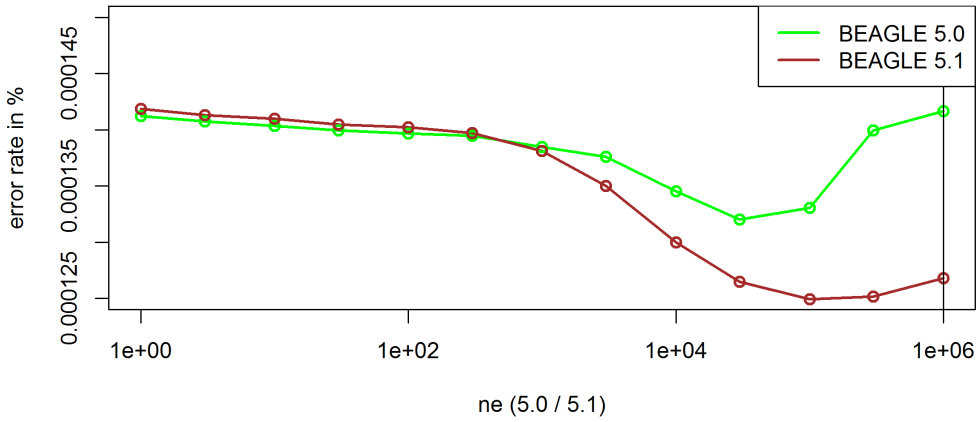


Figure 3.4: Effect of the parameter *ne* on the inference error rates for the maize data in BEAGLE 5.0 and 5.1. Default settings are indicated by the vertical line.

are just for chromosome 10, as not all tests were performed in sufficient sample size for all chromosomes but effect of parameters results should be very similar for all chromosomes. For all three datasets containing heterozygous individuals BEAGLE 5.0 outperformed BEAGLE 5.1, with differences being highest for the set of Pseudo S_0.

3.4.2 Phasing quality

The number of phasing errors for the set of Pseudo S_0 in maize is extremely low with just one phasing error per 2'540 heterozygous markers in BEAGLE 5.1, which should be sufficient for most applications, and the obtainable improvements by parameter tuning were relatively low (Table 3.2). Error rates in BEAGLE 5.0 were about 10% lower (2'716). Biggest improvements in both BEAGLE 5.0 and 5.1 were obtained by adaptation of *ne*. For S_0 the ideal parametrization in BEAGLE 5.0 for phasing is much higher than for inference (Figure 3.35). Especially for BEAGLE 4.0 and 4.1 parameters influencing the structure of the haplotype library had substantial impact on the error rates. In contrast to inference and UM imputation the ideal parametrization for *buildwindow* (4.0) and *phase-states* (5.0 / 5.1) are higher than the default settings (Table 3.10). This in turn leads to only highly related haplotypes to be considered jointly.
To further isolate the structure of phasing errors the same tests were performed on a set of Pseudo S_0 without missing alleles. The interested reader is referred to the Supplementary Table 3.10 for detailed results on this. As phasing is not affected by potential inference errors in this case, error rates are even lower (BEAGLE 5.1 default: one error per 5'756 heterozygous markers, BEAGLE 5.0: 6'141) but the direction of improvement for all parameters stays the same. It should be noted that the maize dataset considered in this study contains highly related individuals

Table 3.2: Inference and phasing error for the 250 Pseudo S_0 lines based on the KE DH-lines for chromosome 10. * BEAGLE crashed for this dataset when using *phase-segment* $>$ 10, *phase-states* $<$ 100 or *phase-states* $>$ 10'000.

Parameter	default	range tested	best inference	best phasing	overall impact
BEAGLE 5.1	-	-	default (0.0239%)	default (2'540)	-
ne	1'000'000	1 - 1'000'000	30 (0.0168%)	10 (3'206)	Figure 3.36
err	0.00015	0.05 - 0.00001	0.0005 (0.0229%)	0.05 (2'556)	Figure 3.38
window	40	10 - 1'000	200 (0.0179%)	200 (2'581)	Figure 3.40
burnin	6	2 - 25	25 (0.0229%)	2 (2'555)	Figure 3.42
iterations	12	2 - 40	default	40 (2'638)	Figure 3.44
phase-states	280	100* - 10'000*	10'000 (0.0168%)	5'000 (2'675)	Figure 3.47
imp-states, imp-segment, cluster, imp-step, imp-nsteps	1'600, 6, 0.005, 0.1, 7	-	-	-	only impacts UM imputation
BEAGLE 5.0	-	-	default (0.0138%)	default (2'716)	-
ne	1'000'000	1 - 1'000'000	1 (0.0111%)	30'000 (3'136)	Figure 3.35
err	0.0001	0.05 - 0.00001	0.005 (0.133%)	0.001 (2'747)	Figure 3.37
window	40	10 - 200	100 (0.0139%)	200 (2'737)	Figure 3.39
burnin	6	2 - 25	2 (0.0136%)	2 (2'748)	Figure 3.41
iterations	12	2 - 40	20 (0.0135%)	40 (2'760)	Figure 3.43
phase-segment	4	1 - 10*	10 (0.132%)	10 (2'758)	Figure 3.45
phase-states	280	100* - 10'000*	10'000 (0.0130%)	5'000 (2'815)	Figure 3.46
imp-states, imp-segment, cluster	1'600, 6, 0.005	-	-	-	only impacts UM imputation
BEAGLE 4.1	-	-	default (0.0345%)	default (2'617)	-
niterations	5	0 - 25	25 (0.0249%)	25 (3'392)	Figure 3.48
modelscale	0.8	0.5 - 5	1 (0.0198%)	1 (3'223)	Figure 3.49
ne	1'000'000	1 - 1'000'000	30 (0.0325%)	30'000 (2'999)	Figure 3.50
BEAGLE 4.0	-	-	default (0.119%)	default (1'240)	-
buildwindow	1'200	1 - 5'000	50 (0.0316%)	5'000 (1'618)	Figure 3.51
singlescale	0.8	0.5 - 5	1.0 (0.0626%)	1.25 (1'955)	Figure 3.52
nsamples	4	1 - 50	50 (0.0780%)	50 (2'308)	Figure 3.53
burnin-its	5	2 - 50	50 (0.108%)	50 (1'599)	Figure 3.54
phase-its	5	2 - 50	50 (0.0944%)	50 (2'320)	Figure 3.55
impute-its	5	2 - 50	-	-	only impacts UM imputation

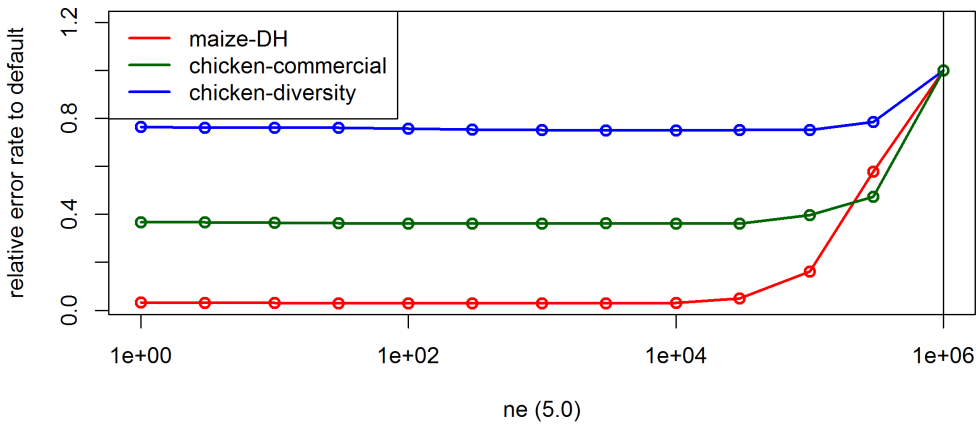


Figure 3.5: Effect of the parameter *ne* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

and a substantial ascertainment bias towards markers with medium allele frequency (Albrechtsen et al., 2010) which both should improve phasing accuracy. For datasets containing less related individuals and sequence data, phasing accuracies can be substantially worse.

3.4.3 UM Imputation quality

The algorithm used for UM imputation in BEAGLE 5.0 and 5.1 is the same, thereby differences are only caused because of slightly different techniques for phasing (B. Browning, personal communication). As no phasing is required for the DH-lines error rates never differed by more than 0.001% and are here reported jointly. When performing UM imputation, error rates were much higher than in the inference case. For all considered datasets tuning of *ne* was absolutely essential (Table 3.3, Figure 3.5), because individuals in the considered livestock and crop datasets are far more related than in an outbred human population with an effective population size of 1'000'000 that is assumed in BEAGLE as default. In the imputation algorithm a low value for *ne* is leading to a reduced probability to switch to a random node in the haplotype cluster and should therefore be beneficial for highly related individuals (Browning and Browning, 2016; Browning et al., 2018). BEAGLE 4.0 does not provide a parameter for the effective population size and is just assuming equidistant markers and fixed switch rates.

All other parameter settings were tested with adapted *ne*, as relative effects were virtually zero otherwise. Appropriate parameter settings for the other parameters were similar to the inference case (Table 3.3) but the overall deviations from the default for *buildwindow*, *singlescale* and *modelscale* were slightly lower. As the number of informative markers in a window with a set number of markers is lower than in the inference case this also makes sense from a modeling perspective. In BEAGLE

Table 3.3: UM imputation error for the KE DH-lines by changing a single imputing parameter with ne = 1'000 for BEAGLE 5.0 / 5.1 and ne = 300 for BEAGLE 4.1.

Parameter	default	range tested	best	overall impact
BEAGLE 5.0 / 5.1	-	-	default (3.09%)	-
ne	1'000'000	1 - 1'000'000	1'000 (0.0877%)	Figure 3.5 and 3.56
err	0.01 (5.0) / 0.00098 (5.1)	0.001 - 0.00001	0.00005 (0.0877%)	Figure 3.57 and 3.58
window	40	10 - 1'000	200 (0.0868%)	Figure 3.59 and 3.60
burnin	6	2 - 25	default (0.0877%)	Figure 3.61 and 3.62
iterations	12	2 - 25	default (0.0877%)	Figure 3.63 and 3.64
phase-segment (5.0)	4	1 - 100	50 (0.0873%)	Figure 3.65
phase-states	280	50 - 1'000	default (0.0877%)	Figure 3.66 and 3.67
imp-states	1'600	100 - 5'000	250 (0.0873%)	Figure 3.68 and 3.69
imp-segment	6	2 - 100	50 (0.0874%)	Figure 3.70 and 3.71
imp-step (5.1)	0.1	0.001 - 20	0.05 (0.0876%)	Figure 3.72
imp-nsteps (5.1)	7	1 - 50	50 (0.875%)	Figure 3.73
cluster	0.005	0.1 - 0.00001	0.00005 (0.0868%)	Figure 3.74 and 3.75
BEAGLE 4.1	-	-	default (6.59%)	-
ne	1'000'000	1 - 1'000'000	300 (0.0958%)	Figure 3.76
niterations	5	0 - 25	-	Figure 3.78
modelscale	0.8	0.5 - 5	2 (0.0886%)	Figure 3.77
BEAGLE 4.0	-	-	default (5.15%)	-
buildwindow	1'200	1 - 2'500	100 (0.799%)	Figure 3.79
singlescale	0.8	0.5 - 5	1.5 (0.188%)	Figure 3.80
nsamples	4	1 - 25	2 (4.36%)	Figure 3.81
burnin-its	5	2 - 50	default	Figure 3.82
phase-its	5	2 - 50	25 (5.071%)	Figure 3.83
impute-its	5	2 - 50	50 (0.189%)	Figure 3.84

5.0 and 5.1 there are additional parameters to control the structure of the haplotype cluster that are only available for UM imputation (*imp-segment*, *imp-states*, *cluster*). Similar to inference, the optimized parameter settings lead to longer and less related haplotypes to be considered jointly. Furthermore, a method to detect identity-by-state (IBS) segments (*imp-step*, *imp-nsteps*) has been added in BEAGLE 5.1 but defaults are already adequately chosen for the maize data. After parameter adaptation error rates in BEAGLE 5.0 and 5.1 were lowest (0.0856% / 0.0857%), followed by BEAGLE 4.1 (0.0887%) and BEAGLE 4.0 (0.139%) (Supplementary Table 3.13).
For both chicken datasets similar results were obtained with BEAGLE 5.0 slightly outperforming BEAGLE 5.1 in for these sets. The interested reader is referred to Supplementary Table 3.8 and 3.9 for detailed results for UM imputation for the chicken panels. Overall, the relative gains by adaptation of *ne* for both the commercial breeding line (0.774% to 0.280% in BEAGLE 5.0) and the diversity chicken panel (3.313% to 2.484% in BEAGLE 5.0) were lower than for the maize data. The optimal parametrization for the effective population for the diversity panel was highest (ne = 3'000). With this, the smaller gains by tuning the effective population size nicely support our expectation of the effective population sizes of the underlying populations. It should still be noted that especially BEAGLE 5.0 and 5.1 were very robust to changes in the effective population size (Figure 3.5 and 3.56) and overall error rates differ by only 0.013% for an effective population size between ne = 1 and ne = 10'000 for the maize dataset, indicating that the use of any reasonable value should work here. As the default of 1'000'000 is not realistic for most livestock and crop datasets, adaptation is necessary and critical when performing UM imputation. For BEAGLE 4.1 there were usually no statistically significant differences between reasonable *ne* values and overall variance in error rates between runs was slightly higher.
As one would expect a larger reference panel leads to smaller error rates for UM imputation (Figure 3.6). Overall, the effect of a larger reference panel in BEAGLE 5.0 was higher than for BEAGLE 4.1. It should still be noted that even for a reference panel with 20 individuals error rates after parameter tuning were below 1% for the maize data and overall error rates only reduce slightly after reaching a size of 150. With higher amounts of overall genetic diversity, the required size of the reference panel should be increasing (Zhang et al., 2013).

3.4.4 Comparison of reference genomes

The most commonly used reference genome in maize genetics is the dent line B73 (Schnable et al., 2009; Jiao et al., 2017). The European maize landraces tested here are considered as flint germplasm with potential major differences in the physical map (Unterseer et al., 2016). After reducing error rates of inference by choosing appropriate parameter settings, markers with high error rates tend to be clustered (Figure 3.7). Markers and regions with high inference error rate can be considered as candidates for misalignment in the genetic map. We compared our results obtained

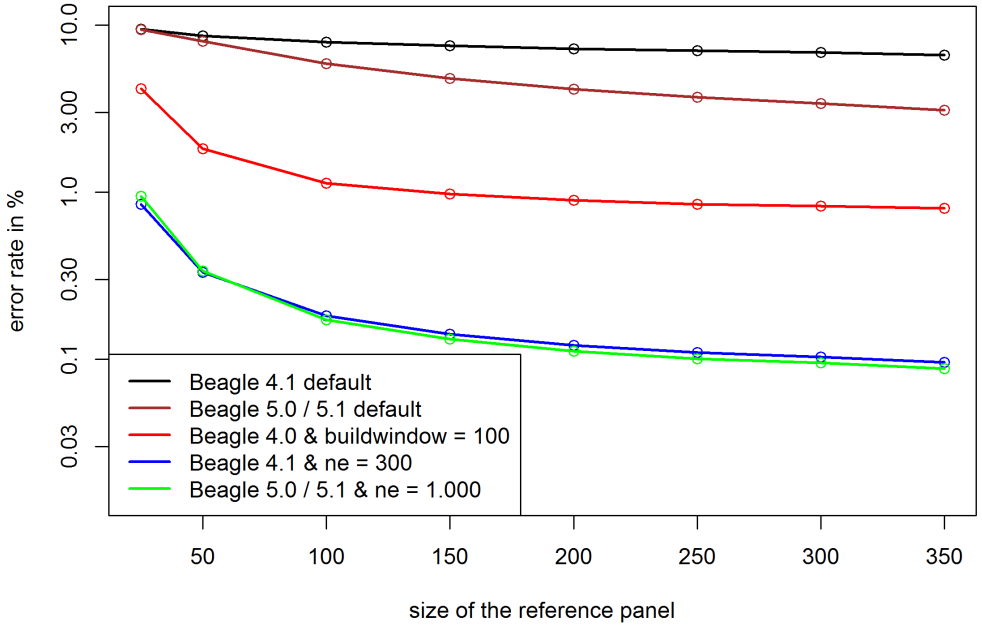


Figure 3.6: Error rates for UM imputation depending on the size of the reference panel in the maize data. Y-axis is log-scaled.

with B73v4 (Jiao et al., 2017) to those obtained with reference genomes of the flint lines F7, EP1, DK105 and PE0075 (Unterseer et al., 2017). Since the array itself was constructed using B73 as a reference (Unterseer et al., 2014) more markers can be mapped to the B73 reference than to the other reference genomes. For those markers mapped to both B73 and the respective flint reference genomes average error rates for inference are reduced by 3-5% (Table 3.4). This improvement is mainly caused by a much reduced number of markers with extremely high error rates. On average, the overall number of markers with error rates above 10% (here referred to as: "critical" markers) is reduced by 57%. For a detailed list of the "critical" markers for all reference genomes mapped on the 600k array (Unterseer et al., 2014), we refer to Supplementary Table 3.5 and 3.6. No notable difference in inference quality for PE when using PE0075 as the reference genome compared to other flint references (Supplementary Table 3.11) was found.

3.4.5 Use of a genetic map

Up to BEAGLE 4.0 all markers are assumed to be equidistant, whereas in BEAGLE 4.1, 5.0 and 5.1 the genetic distance between markers can be provided. On default, the position in base pairs is converted by a ration of 100'000'000 base pairs per Morgan. This might be realistic for human genetics but for chicken a ratio of 41'203'130 / 33'955'860 / 26'631'160 base pairs per Morgan for chromosomes 1 / 7 / 20 is more realistic (Groenen et al., 2009). However, the use of those genetic maps without any further parameter adaptation leads to massively increased error

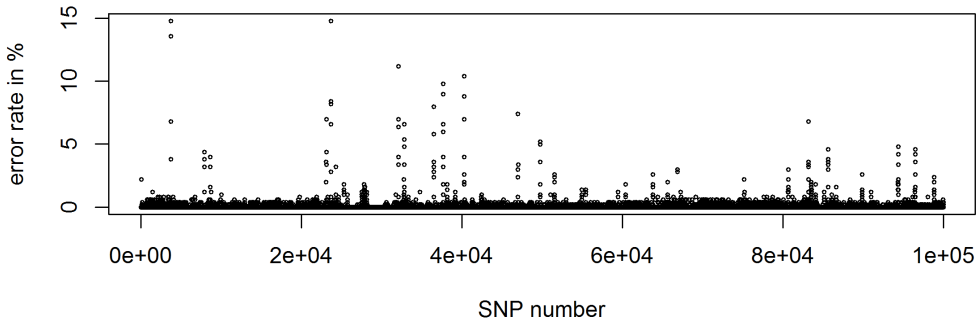


Figure 3.7: Error rate per marker for the first 100'000 SNPs according to physical position (starting with chromosome 1) using BEAGLE 5.0 default with B73v4 (Jiao et al., 2017) as a reference genome.

rates. Error rates for UM imputation increased to 3.23% for the commercial line and 15.8% for the diversity panel compared to the 0.774% and 3.313% without a provided genetic map in BEAGLE 5.0. A potential reason for this is that other parameters like *ne* and *imp-segment* are implicitly affected by the higher distance between markers, leading to smaller segments being considered jointly in the haplotype cluster. After additional fitting of *ne* error rates reduced to values (0.276% / 2.50%) which were very similar to those obtained without use of a genetic map (0.280% / 2.48%; Supplementary Table 3.8 and 3.9).

3.4.6 Quality control using Dosage R-Squared

When performing UM imputation BEAGLE is providing the measurement Dosage R-Squared (DR2; (Browning and Browning, 2009)) as an estimate for the uncertainty for the imputation quality in each respective marker. When using BEAGLE 5.0 with adapted *ne*, only some markers have low DR2 values and the observed error rates in those markers are highly increased (Figure 3.8.A). Markers with DR2 values below 0.8 on average had 140 times as many imputing errors for UM imputation. Note that no scaling for the allele frequency was performed here and no apparent correlation between DR2 values and minor allele frequencies could be observed. In case of no adaption of the effective population size, the number of markers with low DR2 values is massively increased. Even though error rates are still a higher for markers with low DR2, the relative differences are much lower (18 times as many errors for markers with DR2 < 0.8). Even more problematic for filtering is that in contrast to the 44 problematic markers after parameter adaptation, a total of 31'635 of the 62'986 markers in the panel have DR2 values below 0.8 (Figure 3.8.B). Results for the commercial chicken line (Supplementary Figure 3.14) and the diversity panel (Figure Supplementary Figure 3.15) are similar even though differences in DR2 are not as distinct for adapted parameter settings.

Table 3.4: Inference error rates using different reference genomes compared to B73 for KE DH-lines. Only markers mapped on both the flint reference genome & B73v4 (Jiao et al., 2017) are considered for "critical" markers (error rate > 10%).

Reference genome	F7	EP1	DK105	PE0075
Overlapping markers to B73v4	352'326	342'037	338'882	338'244
"Critical" markers when using this map	109	113	115	114
"Critical" markers when using B73v4	271	264	262	262
Relative change in error rate	- 5.11%	-3.87%	-4.68%	-3.32%

3.4.7 Choice of the reference panel

In case the reference population has a lot of stratification, the design of a good reference panel for UM imputation is more difficult, as genetically distant individuals may introduce more noise than relevant information to the model. When comparing results for all considered reference datasets for UM imputation of a single subpopulation it becomes apparent that UM imputation without other individuals from the same subpopulation leads to extremely high error rates (>15%) and thus should in practice only be performed with extreme caution. In contrast, the decision to include other subpopulations in the reference panel is not as clear. When including single other subpopulations in the reference panel we observe significant effects on the overall error rate of UM imputation. Absolute differences of UM imputation error rates are between -0.307% and +0.604% with overall error rates between 1% and 4%. For a detailed list containing all changes in error rates when including a single other subpopulation in the reference panel, we refer to Supplementary Table 3.12. It should be noted that subpopulations with lower genetic distance to the dataset tend to reduce the error rate and a less related subpopulation leads to an increased error rate (Figure 3.9). For all ten subpopulations the slope of the error rate in regard to distance to the subgroup is statistically significantly positive with the main difference between the subpopulations being the intercept. The most extreme case for this is subpopulation 6 (turquoise △ in Figure 3.9; including all wild types). For this group the inclusion of any other subpopulation in the reference panel decreases the imputation quality and is ignored for all averages and statistics

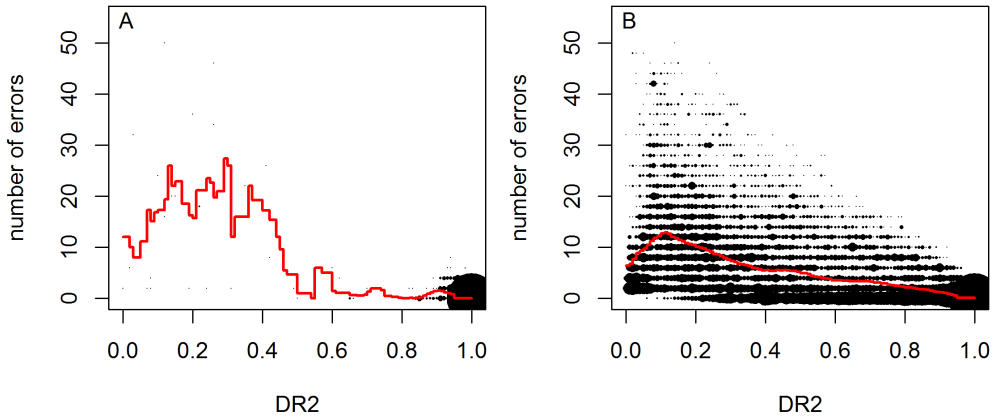


Figure 3.8: DR2 values in relation to the obtained number of error per marker after fitting of *ne* (A) and on default (B) in BEAGLE 5.0 for the maize data. 50 / 350 DH-lines were used for study / reference sample.

in this subsection. Even though SNP-based genetic distances to other subgroups are relatively low, the time to the last common ancestor of any other subpopulation is most likely relatively high. Overall imputation quality when using a reference panel containing all subpopulations is worse than when using a reference panel with only those subpopulation with below average genetic distance (Nei, 1972) to the dataset (2.25% vs. 2.18% - Figure 3.10).
Even though results are statistically significant (two-sample t-test: p-value: 0.0117), differences are only minor and of limited practical relevance for most applications. In our analysis a reference panel containing only the individuals of the same subpopulation on average lead to an UM imputation error of 2.26% with no statistically significant difference to reference panels containing all subpopulations. When performing in-depth analysis for which regions of the dataset UM imputation quality is improved, we observed that especially those individuals with rare variants and overall higher error rates benefited from including more samples in the reference. On the contrary, already well imputed individuals usually had similar or slightly increased error rates. When using a reference panel containing all those subpopulations that individually lead to reduced error rates, average error rates are reduced to 2.06%. It should be noted that a selection based on error rates in UM imputation is usually not possible in practice. Nevertheless, this result demonstrates that there is potential in the use of more sophisticated approaches than just selecting all subpopulation with below average Nei distance (Nei, 1972) as the reference panel. For a detailed list containing error rates for all four different structures of reference panels, we refer to Supplementary Table 3.12.

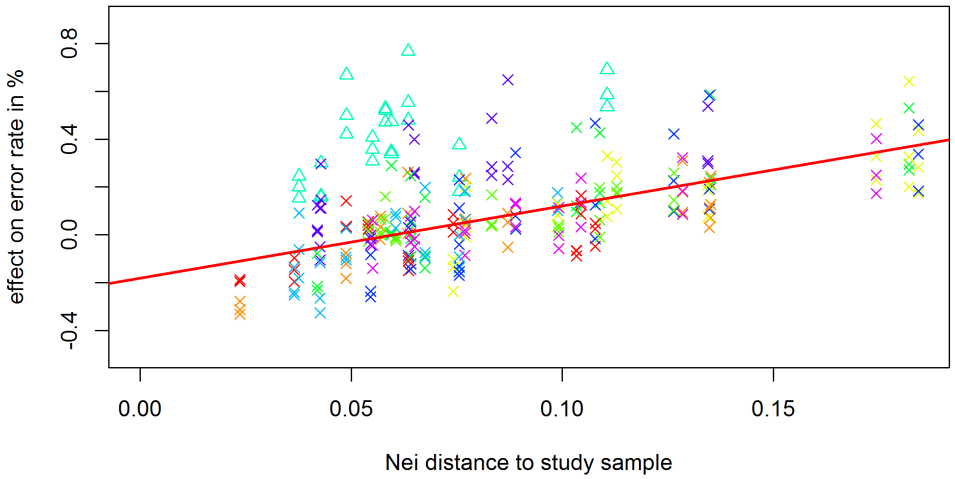


Figure 3.9: Effect of the inclusion of a single subpopulation in the reference panel based on their genetic distance to the dataset for the chicken diversity panel. Colors according to the subpopulation used as the real dataset in Supplementary Figure 3.11. For a detailed list of subpopulation assignment we refer to Supplementary Table 3.7. Subpopulation 6 (including wild types - turquoise △) is ignored in the regression.

3.5 Discussion and Conclusions

3.5.1 Significance of improvement

When comparing error rates under different parameter settings one has to keep in mind the relevance of that optimization. A difference in error rates of 1% in a dataset containing 1% missing genotypes will only result in an improved overall data quality of 0.01% and thus might be negligible compared to other error sources like calling errors (Unterseer et al., 2014). If those improvements would mainly occur in the markers of interest (e.g. markers with low minor allele frequency) or the overall share of missing positions is high (as in UM imputation), this improvement could still be significant for later steps of the analysis.
It should be noted that positions set to NA in this study are chosen at random whereas in a real dataset there might be causal reasons like deletions, leading to some markers with much higher missing rates. When performing imputation on the actual NAs, we observed a higher variance in the imputed allele under different random seeds. As all considered methods always input one of the two allelic variants, this is ignored here but it should be noted that actual error rates are probably a bit higher than reported in this study.

3.5.2 Genetic map and DR2

The used reference genome only mildly affected overall error rates in maize. As the number of markers with extremely high error rates is reduced, we still recommend the use of a reference genome of a more related individual. This of course requires its existence and similar overall quality. The overall gains should not be high enough to justify the generation of a new reference genome just for imputation. Instead one could consider either removing critical markers from the set or use imputation methods like LinkImpute (Money et al., 2015) that do not rely on a genetic map.
We highly recommend the use of DR2 to check validity of results obtained in BEAGLE 5.0 and 5.1. Firstly, observation of a high number of low values of DR2 can be seen as an indicator of overall poor imputation quality. Secondly, one should consider removing markers with low values for DR2 as error rates of UM imputation are typically massively increased. Here, one has to find a balance between removing potentially informative high quality markers and working with low quality markers that could potentially lead to false positive results in later steps of an analysis. In any case, markers that tend to have large effects (e.g. in a genome-wide association study) should be checked for their DR2 value.

3.5.3 Reference panel

Without any knowledge of the genetic structure or excessive testing of genetic relatedness, we recommend to use all available individuals genotyped under high marker density for the reference panel, as the BEAGLE algorithm seems to be quite good at filtering out irrelevant information. However, in case most of the genetic diversity of the study sample can be represented in a subset of the individuals in a reference panel (e.g. a reference panel containing all founder individuals), significant improvements to UM imputation performance can be made by excluding genetically distant individuals. Representing a high share of the genetic diversity of a dataset however is far more important as error rates increase massively if no genomic data of highly related individuals is available in the reference.

3.5.4 Parameter adaptation

Overall, we can conclude that the quality for inference, UM imputation and phasing in BEAGLE 5.0 and 5.1 was better or at least as good as in BEAGLE 4.0 and 4.1 and less tuning of parameters is necessary to obtain good performance for livestock and crop datasets. However, even in BEAGLE 5.0 and 5.1 the adaptation of the parameter *ne* is absolutely necessary when working with genetic datasets with less diversity than a human outbred population. Especially when no parameter tuning in BEAGLE 4.0 and 4.1 was done, one should consider re-running previous preprocessing and quality control protocols. However, a switch from BEAGLE 5.0 to 5.1 is not necessary, nor even recommended as error rates for phasing (and thereby inference and UM imputation) were lower in BEAGLE 5.0. It should be noted

that all datasets in this study contain less genetic diversity than an outbred human population and for datasets with higher genetic diversity like those of UK Biobank (`http://www.ukbiobank.ac.uk/`) BEAGLE 5.1 is supposed to have around 25% lower error rates (B. Browning, personal communications).
Especially for UM imputation and in case of heterozygous individuals an increase of the number of iterations improved results slightly. As long as computing time is no issue we suggest to increase the number of iterations. As the gains by a higher number of iterations is relatively low one can also consider reducing the number of iterations to 4 (or in case of DHs to 2) for large datasets which will dramatically reduce computing time.
Other than in the case of *ne* for UM imputation, improvements in BEAGLE 5.0 and 5.1 by parameter tuning are relatively small, leading us to conclude that the use of default settings should be enough for most applications. Especially for datasets with relatively low genetic diversity one should consider increasing the parameters *phase-segments*, *imp-segments* and *window* while reducing *imp-states* and *ne*. For substantial changes of the imputing parameters and for maximizing the imputing accuracy we strongly suggest to apply a testing pipeline similar to the one suggested in the methods section. As potential gains should not be much higher than 5-10% one has to decide based on the application if this additional effort is worth it. Obtainable improvements in BEAGLE 4.0 and 4.1 are high but we do not recommend to use these versions anymore. Additional benefits of the use of BEAGLE 5.0 and 5.1 are massively reduced computing times and memory requirements. Two potential exceptions to this are if high quality pedigree is available, as only BEAGLE 4.0 is able to incorporate pedigree data in its imputation algorithm and in case only genotype likelihoods are available as input as BEAGLE 5.0 and 5.1 only allow for genotypes as input.

Acknowledgements

The authors thank the German Federal Ministry of Education and Research (BMBF) for the funding of our project (MAZE - “Accessing the genomic and functional diversity of maize to improve quantitative traits”; Funding ID: 031B0195). The "Synbreed - Synergistic Plant and Animal Breeding" project was funded by the German Federal Ministry of Education and Research (FKZ 0315528E).
We also thank Brian Browning for providing quick and thoughtful replies to all our questions regarding insights into BEAGLE and providing us with personalized software updates for BEAGLE 5.1.

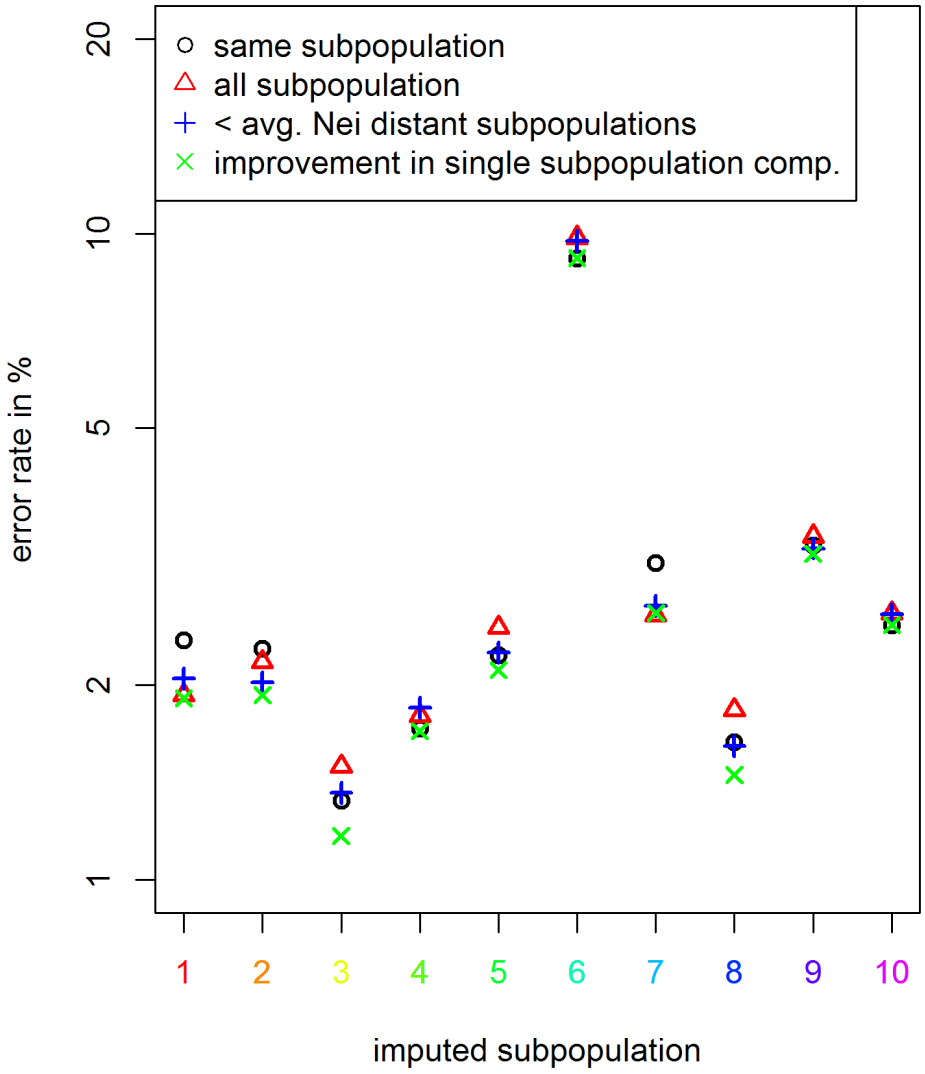


Figure 3.10: Comparison of error rates of UM imputation for different reference panels for the different subpopulations in the chicken diversity panel. Y-axis is log-scaled. For a detailed list on which individual is assigned to which subpopulation we refer to Supplementary Table 3.7.

3.6 Supplementary Material

The numbering of the Supplementary material corresponds to the order in which they are listed in this section.

3.6.1 Supplementary tables

As both Table S2 and Table S3 of the submitted manuscript contain lists of the "critical" markers for Kemater Landmais Gelb and Petkuser Ferdinand Rot for the different reference panels (B73, F7, EP1, DK105, PE0075). As both lists contain more than 700 SNPs each, they are not displayed here. For similar reason Table S8 that is containing the subpopulation assignment for each of the 1'810 chicken is not displayed here. Table captions are still provided in Table 3.5, 3.6 and 3.7. The interested reader is referred to FigShare (`https://gsajournals.figshare.com/s/4c6ceaa7fe834f0700a3`) for the complete files.

Table 3.5: List of "critical" markers for Kemater Landmais Gelb using different reference genomes.

Table 3.6: List of "critical" markers for Petkuser Ferdinand Rot using different reference genomes.

Table 3.7: Assignments to subpopulations for the chicken diversity panel based on Nei standard genetic distances (Nei, 1972).

Table 3.8: UM imputation error for the commercial breeding line in chicken by changing a single imputing parameter with $ne = 100$ for BEAGLE 5.1, $ne = 300$ for BEAGLE 5.0 and $ne = 10'000$ for BEAGLE 4.1. * BEAGLE crashed for this dataset when using *phase-segment* > 10, *phase-states* < 100 or *imp-step* > 5.

Parameter	default	range tested	best	overall impact
BEAGLE 5.1	-	-	default (0.774%)	-
ne	1'000'000	1 - 1'000'000	100 (0.282%)	Figure 3.56
err	0.00043	0.01 - 0.00001	0.001 (0.280%)	Figure 3.58
window	40	5 - 200	10 (0.260%)	Figure 3.60
burnin	6	2 - 50	50 (0.282%)	Figure 3.62
iterations	12	2 - 50	50 (0.282%)	Figure 3.64
phase-states	280	100* - 1'000	default	Figure 3.67
imp-states	1'600	100 - 5'000	default	Figure 3.69
imp-segment	6	2 - 100	20 (0.282%)	Figure 3.71
imp-step	0.1	0.01 - 5*	5 (0.270%)	Figure 3.72
imp-nsteps	7	1 - 50	50 (0.282%)	Figure 3.73
cluster	0.005	0.1 - 0.0001	default	Figure 3.75
BEAGLE 5.0	-	-	default (0.774%)	-
ne	1'000'000	1 - 1'000'000	300 (0.280%)	Figure 3.5
err	0.0001	0.01 - 0.00001	0.005 (0.276%)	Figure 3.57
window	40	5 - 200	10 (0.267%)	Figure 3.59
burnin	6	2 - 50	50 (0.279%)	Figure 3.61
iterations	12	2 - 50	50 (0.279%)	Figure 3.63
phase-segment	4	1-10*	10 (0.279%)	Figure 3.65
phase-states	280	100* - 1'000	500 (0.280%)	Figure 3.66
imp-states	1'600	100 - 5'000	1'000 (0.278%)	Figure 3.68
imp-segment	6	2 - 100	20 (0.280%)	Figure 3.70
cluster	0.005	0.1 - 0.0001	default	Figure 3.74
BEAGLE 4.1	-	-	default (0.955%)	-
ne	1'000'000	1 - 1'000'000	30'000 (0.277%)	Figure 3.76
niterations	5	0 - 25	50 (0.276%)	Figure 3.78
modelscale	0.8	0.5 - 5	1.5 (0.276%)	Figure 3.77
BEAGLE 4.0	-	-	default (0.286%)	-
buildwindow	1'200	1 - 2'500	default	Figure 3.79
singlescale	0.8	0.5 - 5	1.25 (0'281%)	Figure 3.80
nsamples	4	1 - 25	25 (0.276%)	Figure 3.81
burnin-its	5	2 - 50	50 (0.282%)	Figure 3.82
phase-its	5	2 - 50	50 (0.278%)	Figure 3.83
impute-its	5	2 - 50	50 (0.268%)	Figure 3.84

Table 3.9: UM imputation error for the diversity panel in chicken by changing a single imputing parameter with $ne = 300$ for BEAGLE 5.1, $ne = 3'000$ for BEAGLE 5.0 and $ne = 10'000$ for BEAGLE 4.1. * BEAGLE crashed for this dataset when using *phase-segment* > 10 or *phase-states* < 100.

Parameter	default	range tested	best	overall impact
BEAGLE 5.1	-	-	default (3.399%)	-
ne	1'000'000	1 - 1'000'000	300 (2.572%)	Figure 3.56
err	0.00043	0.01 - 0.00001	0.001 (2.542%)	Figure 3.58
window	40	5 - 200	5 (2.543%)	Figure 3.60
burnin	6	2 - 50	50 (2.568%)	Figure 3.62
iterations	12	2 - 50	50 (2.549%)	Figure 3.64
phase-states	280	10* - 1'000	1'000 (2.552%)	Figure 3.67
imp-states	1'600	100 - 5'000	250 (2.552%)	Figure 3.69
imp-segment	6	2 - 100	50 (2.542%)	Figure 3.71
imp-step	0.1	0.005 - 5	0.01 (2.550%)	Figure 3.72
imp-nsteps	7	1 - 50	2 (2.569%)	Figure 3.73
cluster	0.005	0.1 - 0.0001	default	Figure 3.75
BEAGLE 5.0	-	-	default (3.313%)	-
ne	1'000'000	1 - 1'000'000	3'000 (2.484%)	Figure 3.5
err	0.0001	0.01 - 0.00001	0.005 (2.454%)	Figure 3.57
window	40	10 - 200	10 (2.479%)	Figure 3.59
burnin	6	2 - 50	50 (2.464%)	Figure 3.61
iterations	12	2 - 50	50 (2.472%)	Figure 3.63
phase-segment	4	1-10*	10 (2.457%)	Figure 3.65
phase-states	280	10* - 1'000	100 (2.475%)	Figure 3.66
imp-states	1'600	100 - 5'000	250 (2.483%)	Figure 3.68
imp-segment	6	2 - 100	50 (2.471%)	Figure 3.70
cluster	0.005	0.1 - 0.0001	default	Figure 3.74
BEAGLE 4.1	-	-	default (3.695%)	-
ne	1'000'000	1 - 1'000'000	1'000 (2.722%)	Figure 3.76
niterations	5	0 - 50	50 (2.592%)	Figure 3.78
modelscale	0.8	0.5 - 5	default	Figure 3.77
BEAGLE 4.0	-	-	default (4.207%)	-
buildwindow	1'200	1 - 2'500	50 (3.962%)	Figure 3.79
singlescale	0.8	0.5 - 5	default	Figure 3.80
nsamples	4	1 - 25	25 (3.892%)	Figure 3.81
burnin-its	5	2 - 50	50 (4.169%)	Figure 3.82
phase-its	5	2 - 50	50 (3.928%)	Figure 3.83
impute-its	5	2 - 50	50 (3.820%)	Figure 3.84

Table 3.10: Phasing error, as number of heterozygous markers per switch error, for Pseudo S_0 generated based on the KE DH-lines by changing a single imputing parameter.

Parameter	default	range tested	best
BEAGLE 5.1	-	-	default (5'756)
ne	1'000'000	1 - 1'000'000	1 (8'171)
err	0.0001	0.001 - 0.00001	0.0005 (5'768)
window	40	10 - 1'000	200 (6'147)
burnin	6	2 - 50	default
iterations	12	2 - 40	40 (5'938)
phase-states	280	100 - 10'000	10'000 (6'227)
imp-states, imp-segment, cluster, imp-step, imp-nsteps	1'600, 6, 0.005, 0.1, 7	-	only impacts UM imputation
BEAGLE 5.0	-	-	default (6'141)
ne	1'000'000	1 - 1'000'000	10'000 (9'008)
err	0.0001	0.001 - 0.00001	0.005 (6'203)
window	40	10 - 200	default
burnin	6	2 - 50	2 (6'412)
iterations	12	2 - 40	40 (6'239)
phase-segment	4	1 - 10	10 (6'805)
imp-states, imp-segment, cluster	1'600, 6, 0.005	-	only impacts UM imputation
BEAGLE 4.1	-	-	default (5'876)
niterations	5	0 - 50	50 (6'651)
modelscale	0.8	0.5 - 5	1.25 (10'643)
ne	1'000'000	1 - 1'000'000	-
BEAGLE 4.0	-	-	default (2'466)
buildwindow	1'200	1 - 2'500	2'500 (2'588)
singlescale	0.8	0.5 - 5	1.25 (3'624)
nsamples	4	1 - 50	50 (4'392)
burnin-its	5	2 - 50	50 (3'082)
phase-its	5	2 - 50	25 (2'644)
impute-its	5	2 - 50	50 (3'228)

Table 3.11: Inference error rates using different reference genomes compared to B73 for PE DH-lines. Only markers mapped on both the flint reference genome & B73v4 (Jiao et al., 2017) are considered for "critical" markers (error rate > 10%).

Reference genome	F7	EP1	DK105	PE0075
Overlapping markers to B73v4	357'757	357'191	353'834	352'514
"Critical" markers when using this map	96	118	114	107
"Critical" markers when using B73v4	239	236	233	233
Relative change in error rate	- 6.94%	-0.98%	-3.84%	-1.90%

Table 3.12: Error rates for UM imputation for different reference panels, including A (same subpopulation), C (all other subpopulation), D (below average Nei distant subpopulations), E (All subpopulation with reduced error rate when testing A + B compared to A as the reference panel).

Subpopulation	A	A + C	A + D	A + E	Population in E
1	2.345%	1.932%	2.050%	1.909%	2, 5, 7, 9
2	2.280%	2.172%	2.020%	1.931%	1, 7
3	1.324%	1.498%	1.352%	1.168%	1
4	1.713%	1.794%	1.846%	1.697%	7
5	2.225%	2.456%	2.245%	2.110%	7, 8, 9
6	9.162%	9.858%	9.737%	9.162%	-
7	3.089%	2.571%	2.652%	2.587%	1, 2, 6, 8, 9
8	1.630%	1.831%	1.609%	1.452%	5, 7, 9
9	3.295%	3.400%	3.256%	3.194%	7, 8
10	2.474%	2.585%	2.574%	2.479%	4, 6, 7, 9

Table 3.13: Minimal obtained error rates and used parameter settings for inference and UM imputation. Deviations from the ideal single parameter settings are caused by BEAGLE crashing when changing parameters jointly.

Dataset	Imputation type	BEAGLE version	Error rate	Parameter settings
Maize DH-lines	Inference	4.0	0.0307%	buildwindow = 25, nsamples = 25
Maize DH-lines	Inference	4.1	0.0436%	modelscale = 1.5, ne = 10'000
Maize DH-lines	Inference	5.0	0.0132%	window = 150, ne = 10'000, phase-segment = 6, phase-states = 150
Maize DH-lines	Inference	5.1	0.0122%	window = 200, ne = 100'000, err = 0.001
Maize Pseudo S_0	Inference	4.0	0.0193% (2'670)	buildwindow = 50, nsamples = 50, burnin-its = 50, phase-its = 50
Maize Pseudo S_0	Inference	4.1	0.0168% (3'563)	modelscale = 1, ne = 30, niterations = 25
Maize Pseudo S_0	Inference	5.0	0.0109% (3'429)	ne = 1, err = 0.0005, window = 100, burnin = 2, phase-states = 5'000
Maize Pseudo S_0	Inference	5.1	0.0148% (3'438)	ne = 30, err = 0.005, window = 200, burnin = 2, phase-states = 25'000
Maize Pseudo S_0	Phasing	4.0	0.0644% (4'180)	buildwindow = 5'000, nsamples = 50, burnin-its = 50, phase-its = 50
Maize Pseudo S_0	Phasing	4.1	0.0177% (3'684)	modelscale = 1, ne = 30'000, niterations = 25
Maize Pseudo S_0	Phasing	5.0	0.0109% (3'457)	ne = 30'000, err = 0.001, window = 200, burnin = 2, phase-states = 500, iterations = 40
Maize Pseudo S_0	Phasing	5.1	0.0159% (3'524)	ne = 30, err = 0.05, window = 200, burnin = 2, phase-states = 5'000, iterations = 40
Maize DH-lines	UM imputation	4.0	0.139%	singlescale = 1.5, impute-its = 50, phase-its = 50
Maize DH-lines	UM imputation	4.1	0.0887%	modelscale = 1.5, ne = 300
Maize DH-lines	UM imputation	5.0	0.0856%	ne = 1'000, window = 200, phase-segment = 50, imp-states = 250, imp-segment = 50, cluster = 0.00005
Maize DH-lines	UM imputation	5.1	0.0857%	ne = 1'000, window = 200, imp-nsteps= 50, imp-states = 250, imp-segment = 50, cluster = 0.00005
Commercial chicken	UM imputation	4.0	0.262%	nsamples=25, singlescale = 1.25, burnin-its = 50, phase-its = 50, impute-its = 50
Commercial chicken	UM imputation	4.1	0.274%	ne = 30'000, niterations = 50, modelscale = 1.5
Commercial chicken	UM imputation	5.0	0.261%	ne = 300, err = 0.005 window = 10, burnin = 50, iterations = 50, phase-segment = 10, imp-states = 1'000
Commercial chicken	UM imputation	5.1	0.252%	ne = 100, err = 0.001, window = 10, burnin = 50, iterations = 50, imp-step=5, imp-nsteps=50
Chicken diversity panel	UM imputation	4.0	3.302%	buildwindow = 50, nsamples = 25, burnin-its = 50, phase-its = 50, impute-its = 50
Chicken diversity panel	UM imputation	4.1	2.592%	ne = 1'000, niterations = 50
Chicken diversity panel	UM imputation	5.0	2.409%	ne = 3'000, err = 0.005, window = 10, burnin = 50, iterations = 50, phase-segment = 8, phase-states = 125, imp-segment = 50
Chicken diversity panel	UM imputation	5.1	2.525%	ne = 300, err = 0.001, window = 20, burnin = 50, iterations = 50, imp-states = 800, imp-segment = 10, imp-nsteps = 50

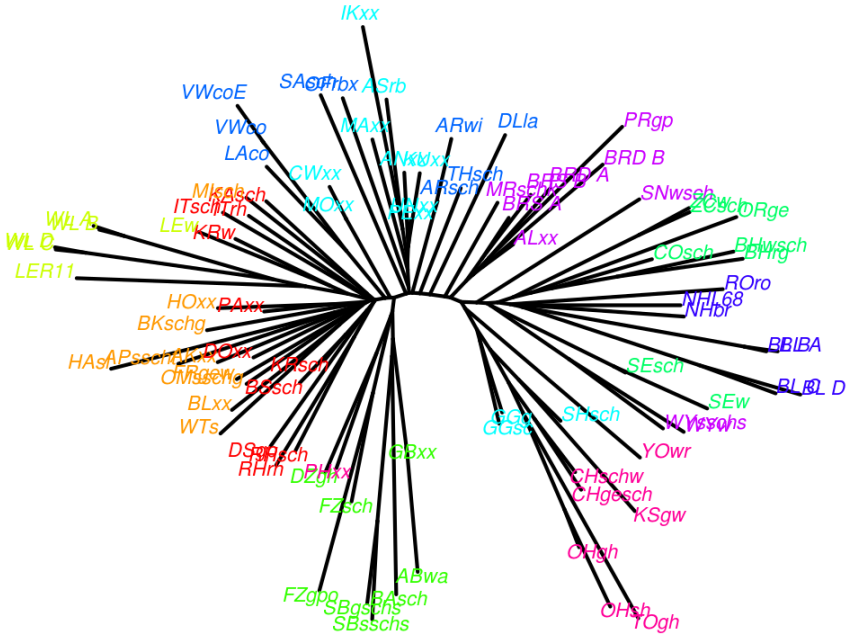


Figure 3.11: Neighbor-joining-tree for ten subpopulations in the chicken diversity panel. For a detailed list on which individual is assigned to which subpopulation we refer to Supplementary Table 3.7.

3.6.2 Supplementary figures

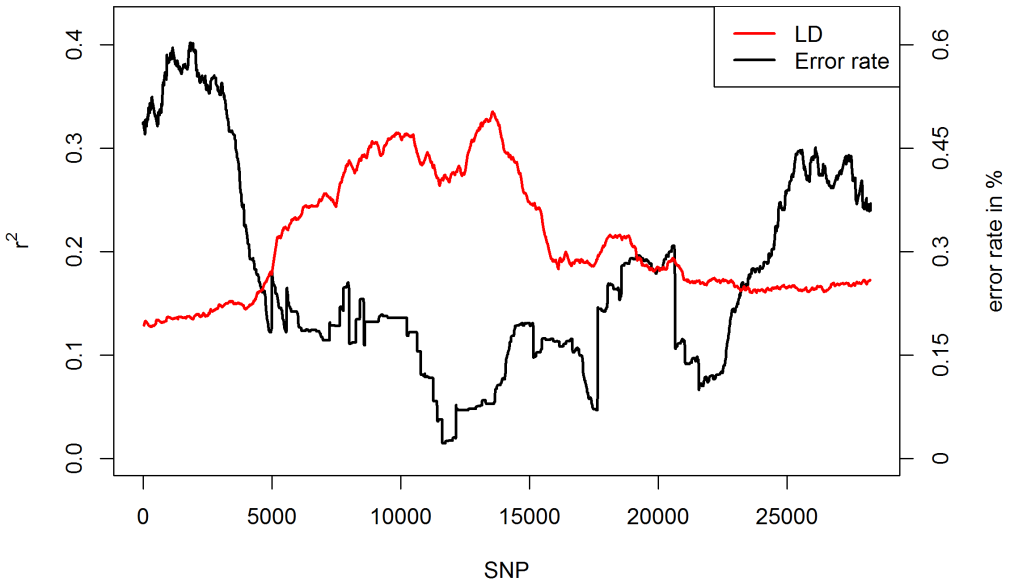


Figure 3.12: Relationship between region error rate and LD (r^2) on chromosome 9 in the maize data. Outliers are corrected for by using a Nadaraya-Watson-estimator (Nadaraya, 1964), using a Gaussian kernel and a bandwidth of 3'000 markers in both cases.

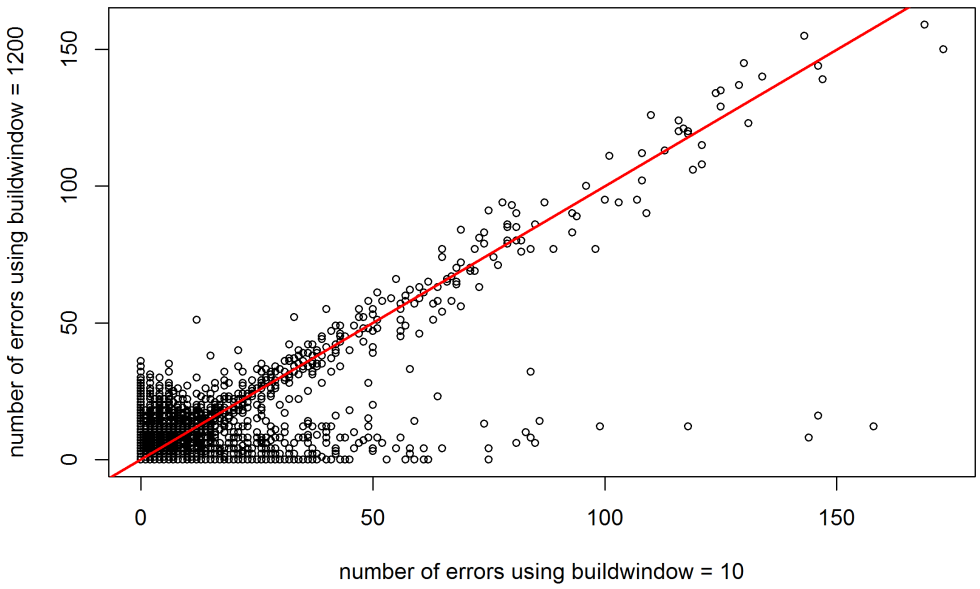


Figure 3.13: Total number of errors per marker (50 repetitions) for BEAGLE 4.0 using *buildwindow* of 10 and 1200 (default) in the maize data.

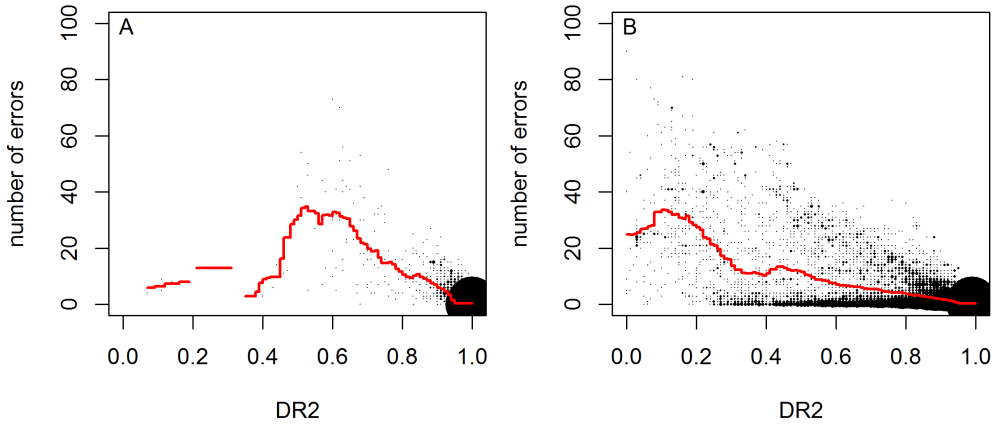


Figure 3.14: DR2 values in relation to the obtained number of error per marker after fitting of *ne* (A) and on default (B) in BEAGLE 5.0 for the commercial chicken line. 100 / 788 lines were used for study / reference sample.

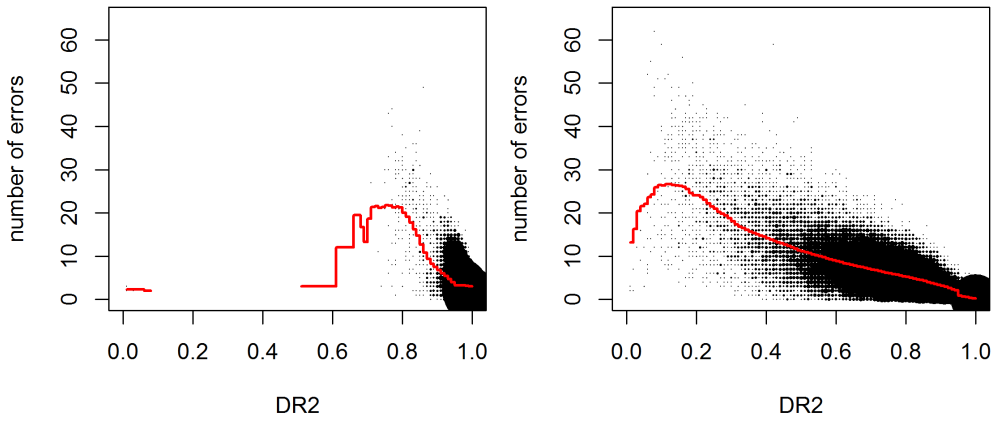


Figure 3.15: DR2 values in relation to the obtained number of error per marker after fitting of *ne* (A) and on default (B) in BEAGLE 5.0 for the chicken diversity panel. 100 / 1710 lines were used for study / reference sample.

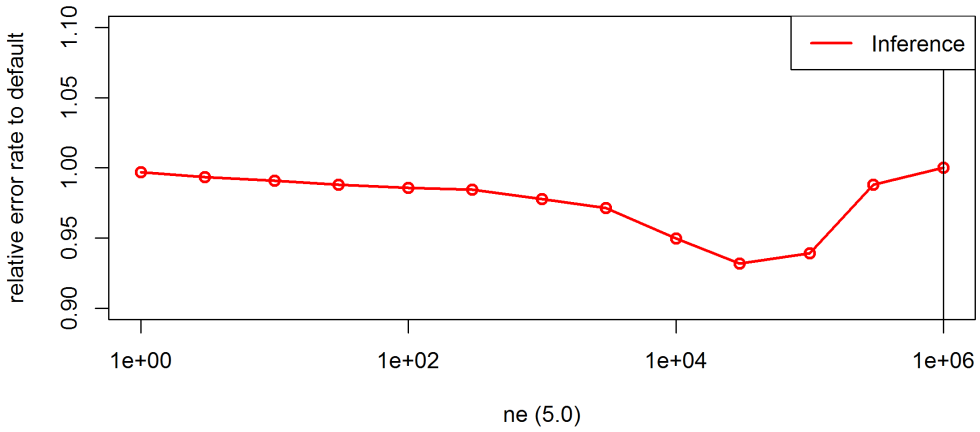


Figure 3.16: Effect of the parameter *ne* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

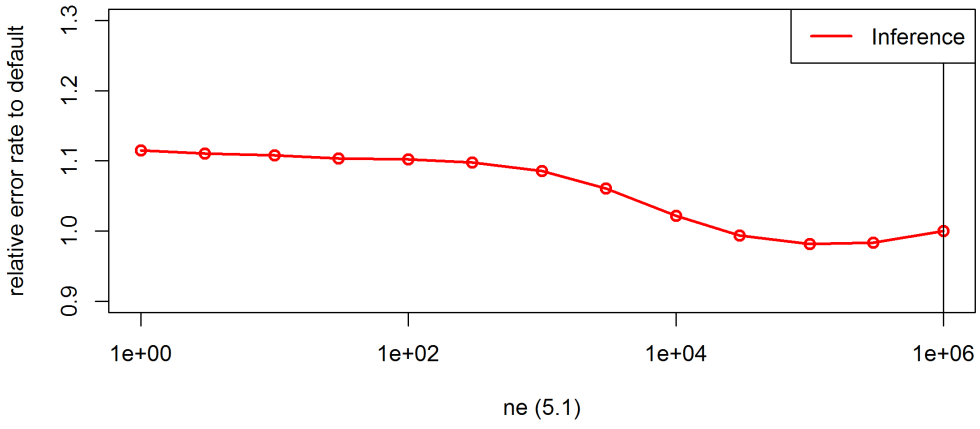


Figure 3.17: Effect of the parameter *ne* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

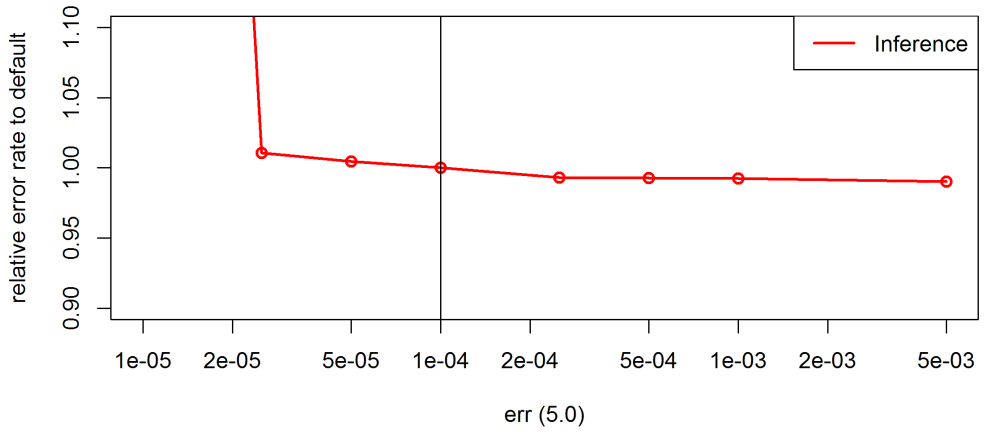


Figure 3.18: Effect of the parameter *err* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

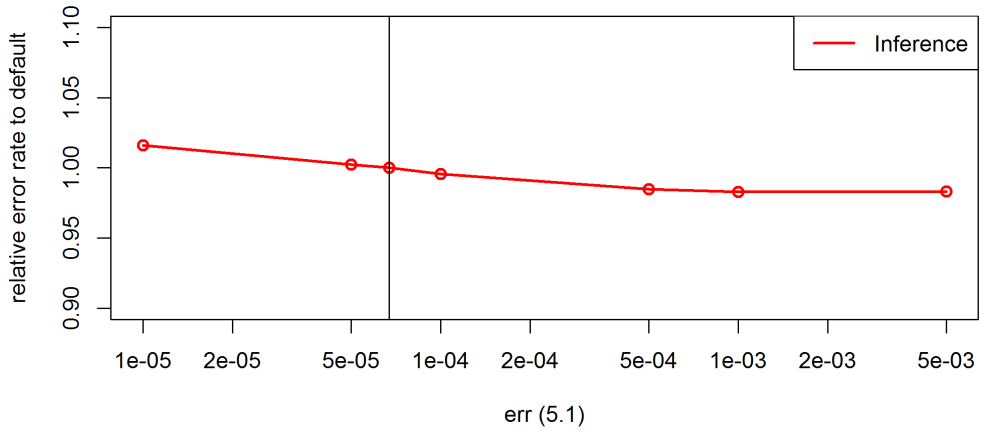


Figure 3.19: Effect of the parameter *err* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

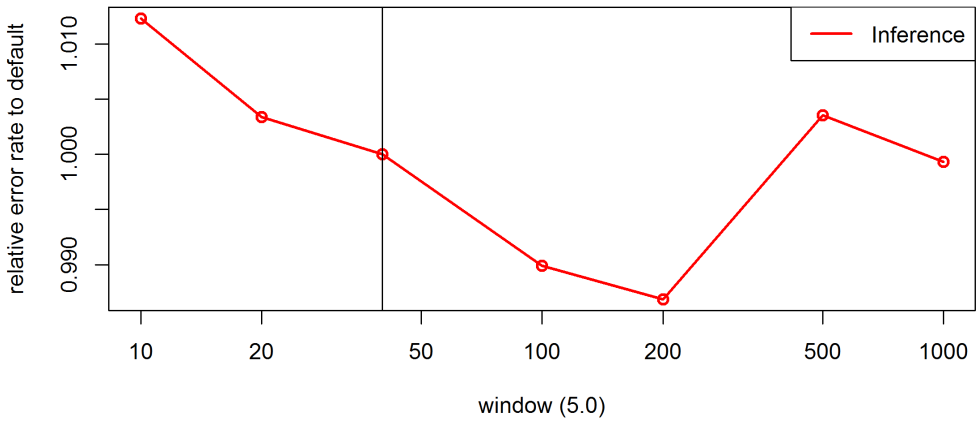


Figure 3.20: Effect of the parameter *window* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

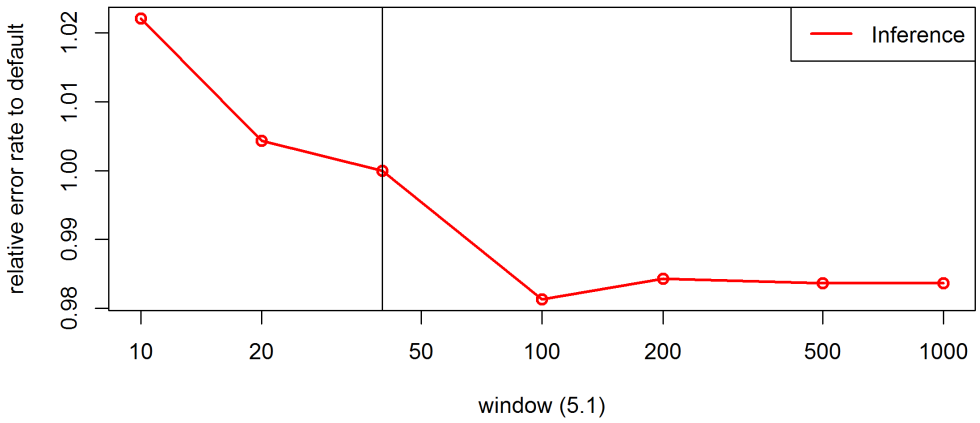


Figure 3.21: Effect of the parameter *window* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

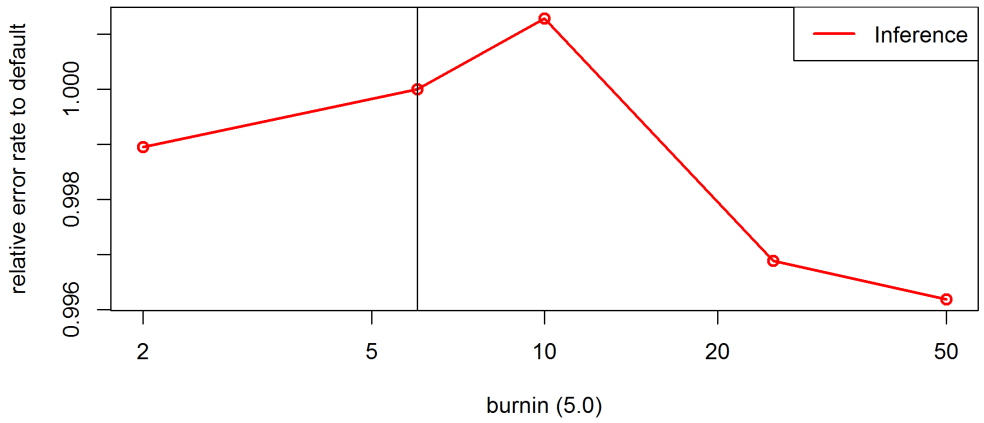


Figure 3.22: Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

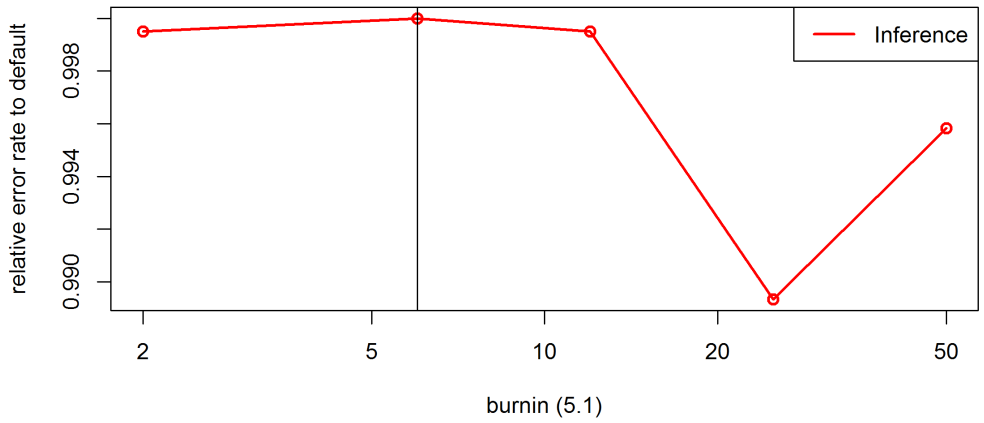


Figure 3.23: Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

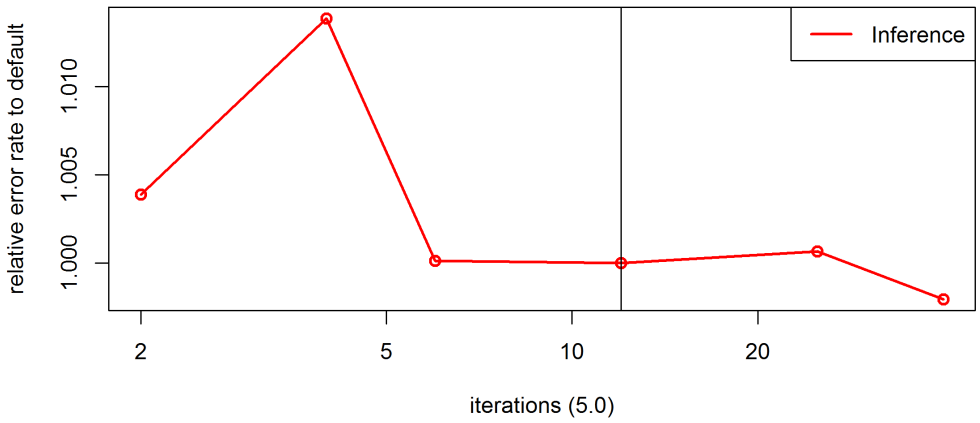


Figure 3.24: Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

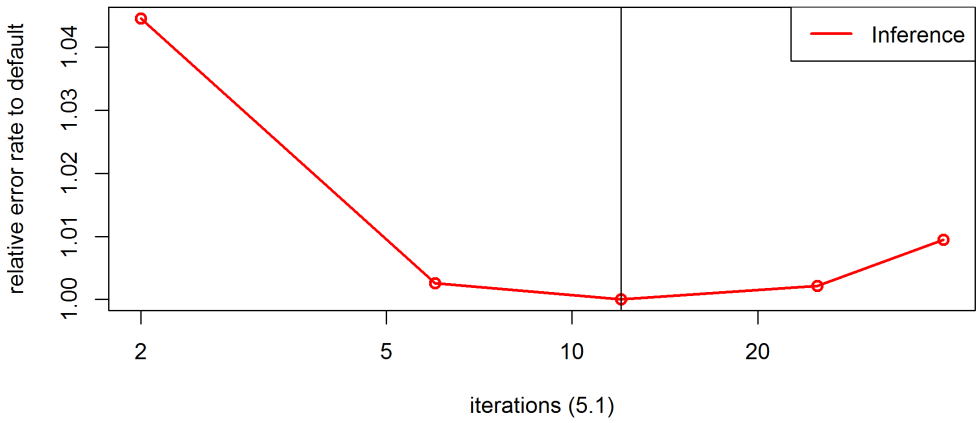


Figure 3.25: Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

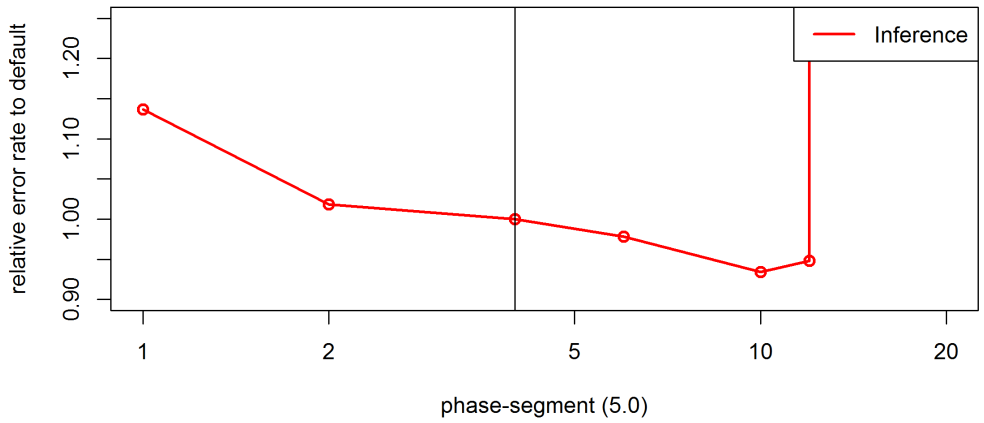


Figure 3.26: Effect of the parameter *phase-segment* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

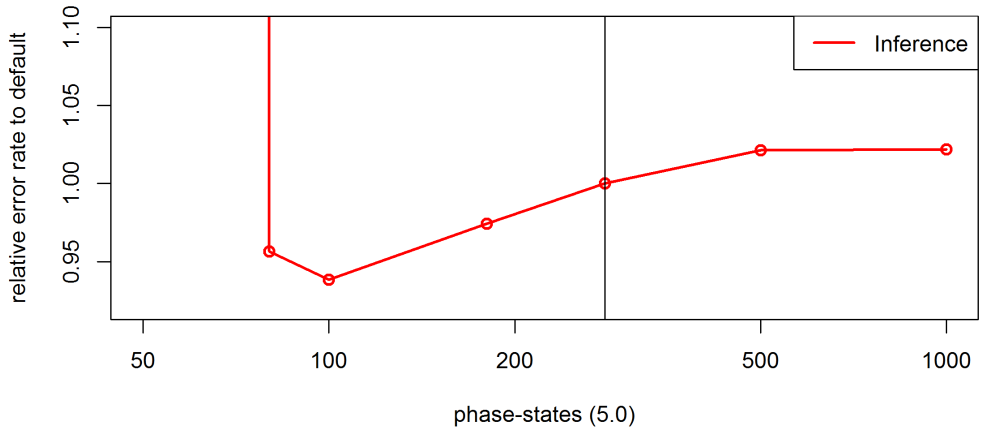


Figure 3.27: Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

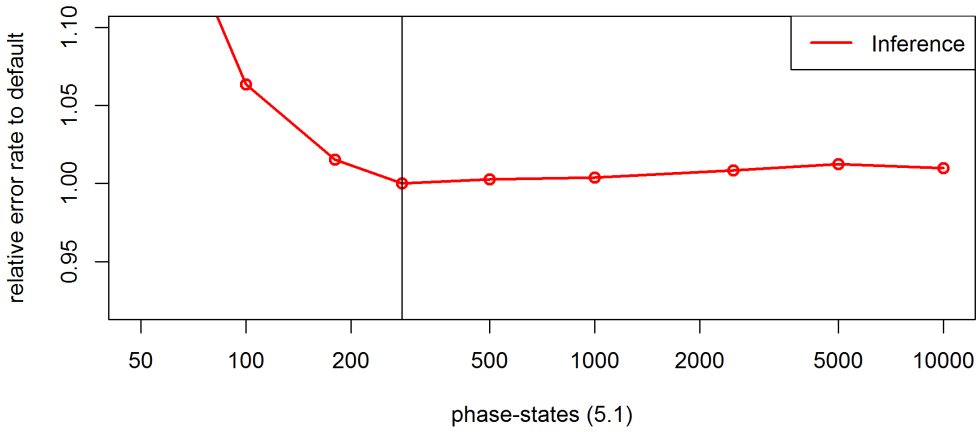


Figure 3.28: Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

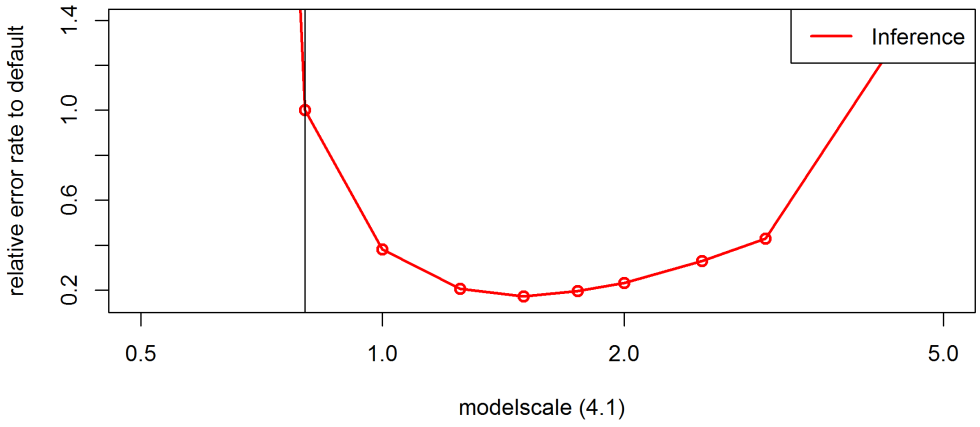


Figure 3.29: Effect of the parameter *modelscale* on the inference error rates for the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line.

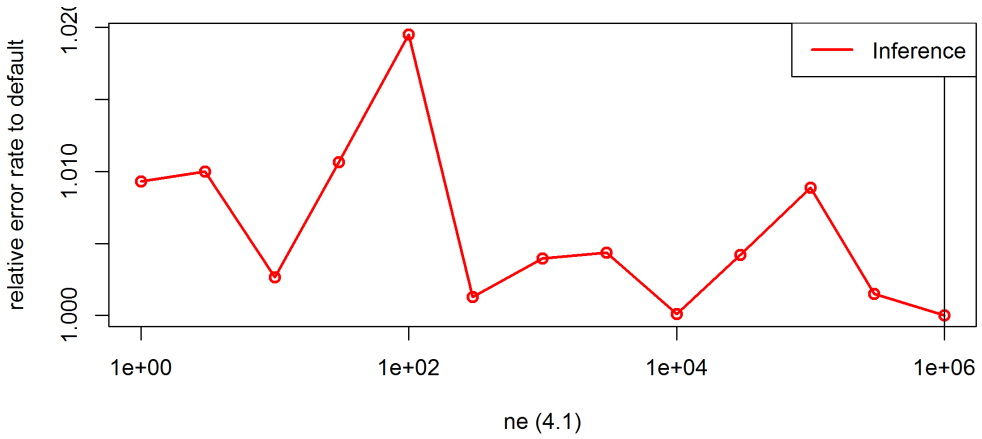


Figure 3.30: Effect of the parameter *ne* on the inference error rates for the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line.

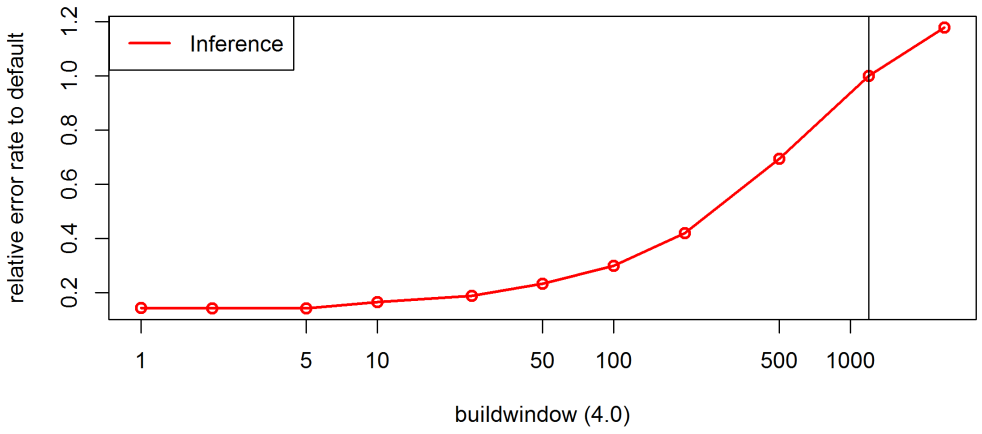


Figure 3.31: Effect of the parameter *buildwindow* on the inference error rates for the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

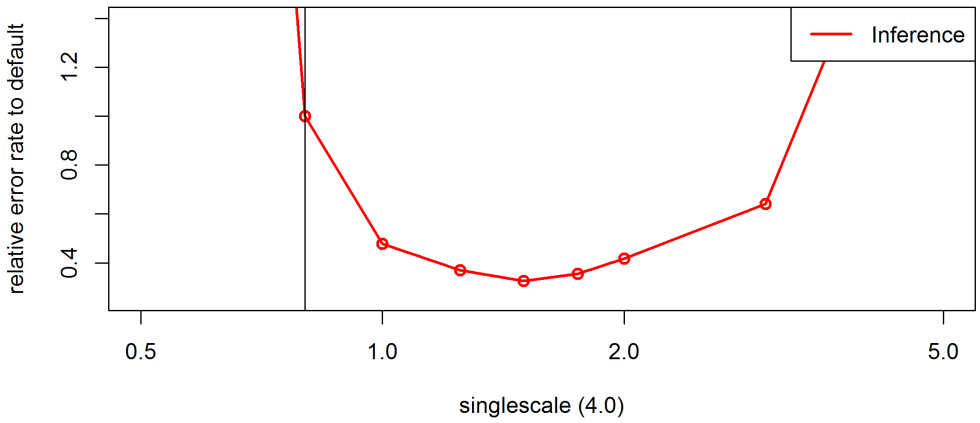


Figure 3.32: Effect of the parameter *singlescale* on the inference error rates for the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

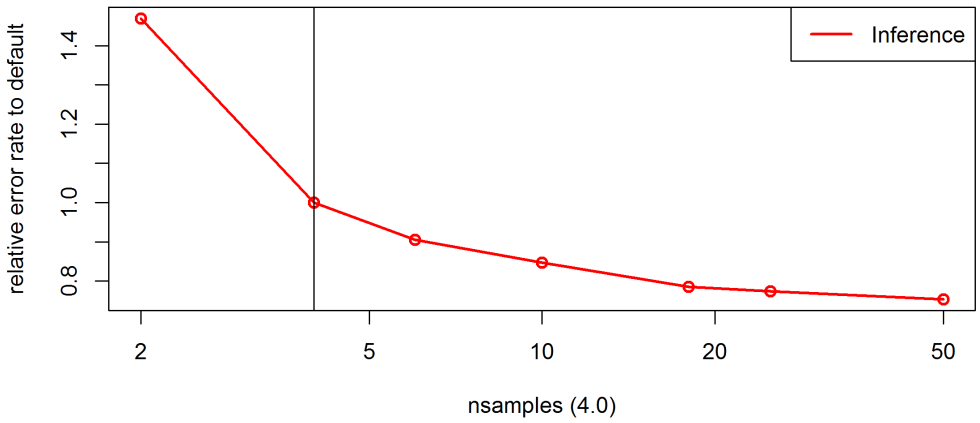


Figure 3.33: Effect of the parameter *nsamples* on the inference error rates for the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

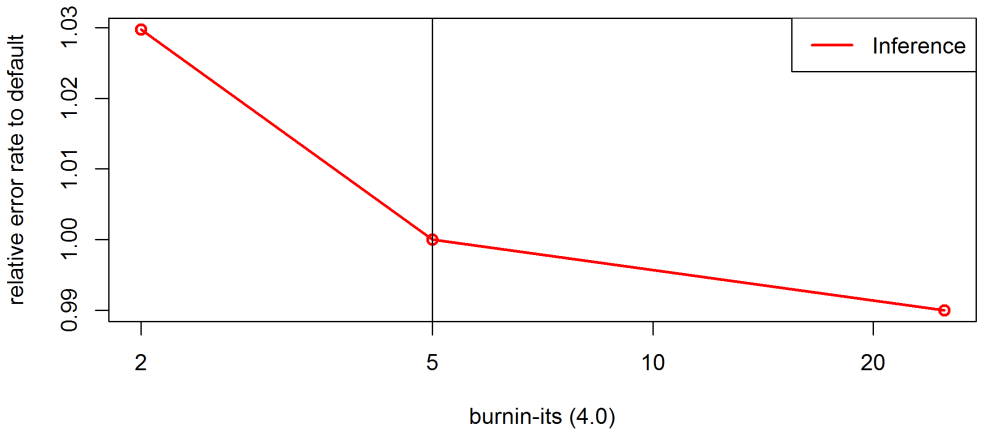


Figure 3.34: Effect of the parameter *burnin-its* on the inference error rates for the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

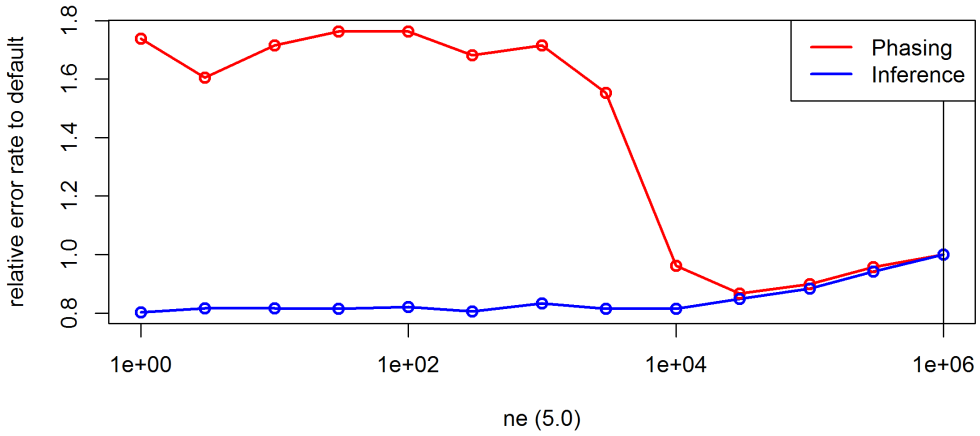


Figure 3.35: Effect of the parameter *ne* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

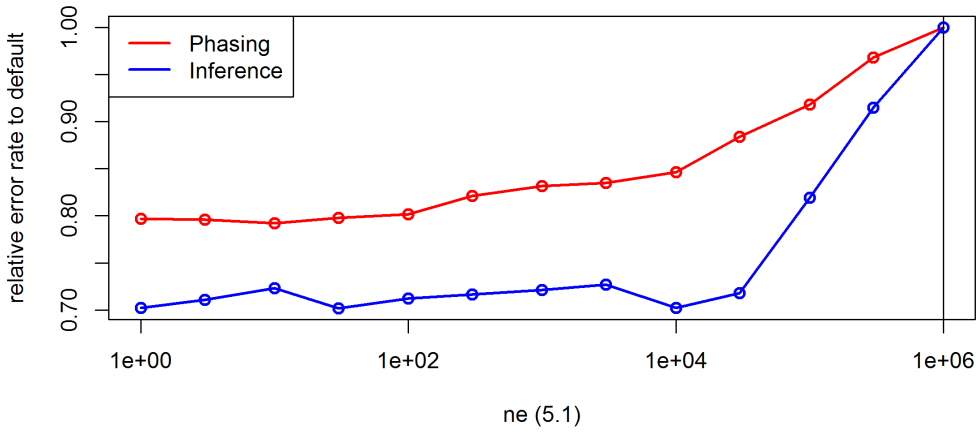


Figure 3.36: Effect of the parameter *ne* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

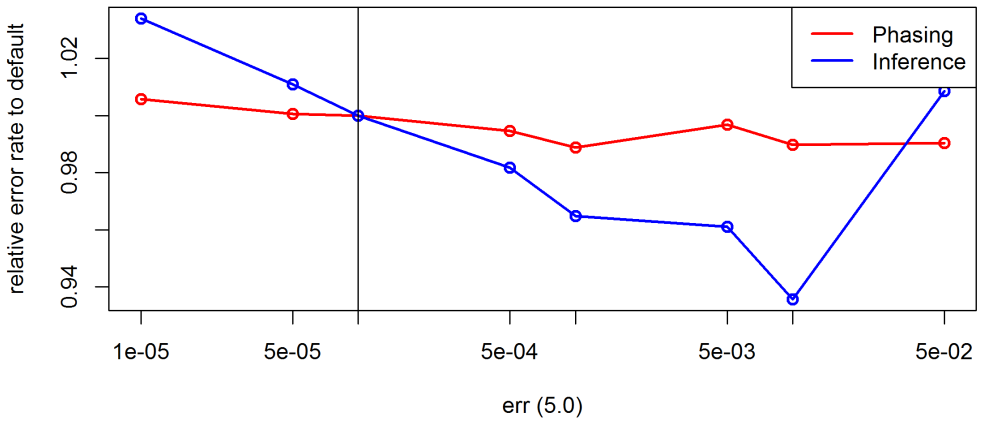


Figure 3.37: Effect of the parameter *err* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

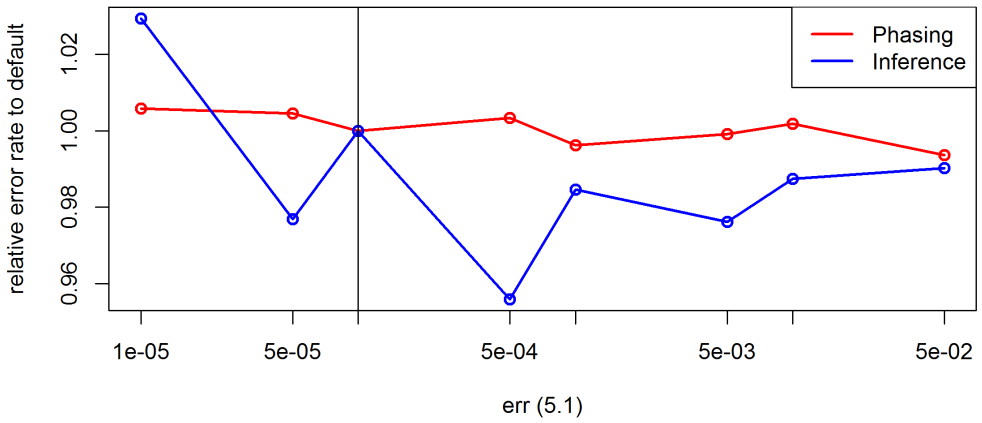


Figure 3.38: Effect of the parameter *err* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

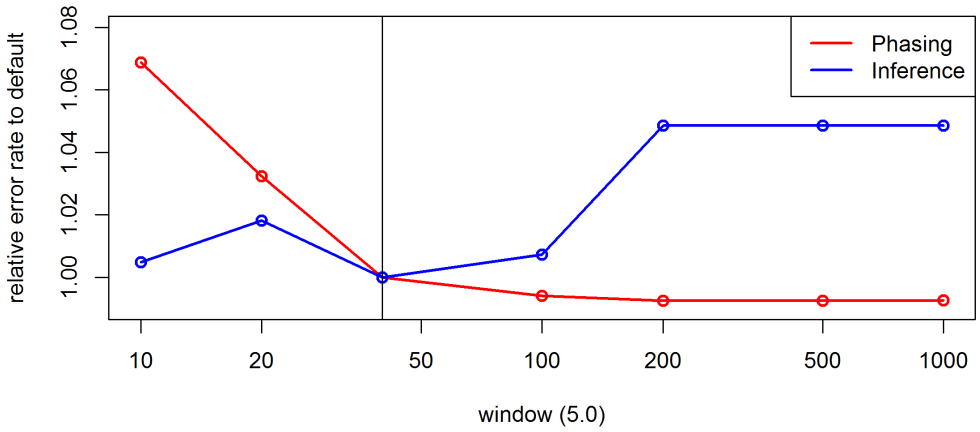


Figure 3.39: Effect of the parameter *window* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

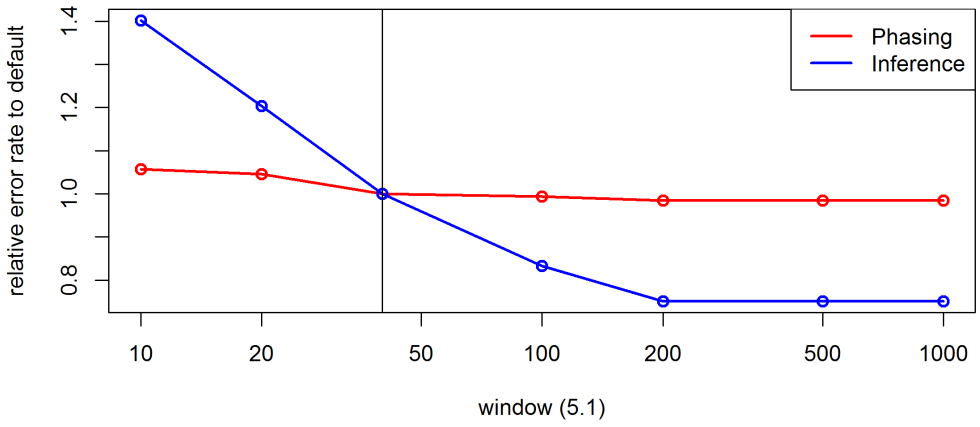


Figure 3.40: Effect of the parameter *window* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

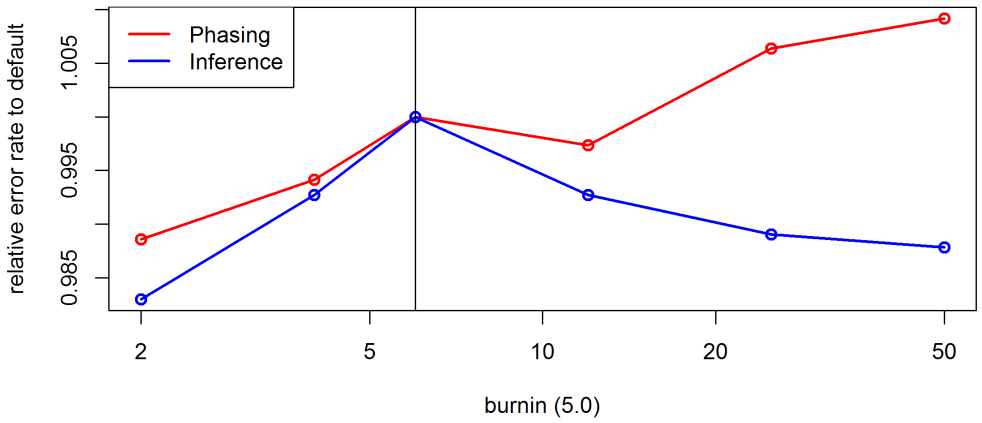


Figure 3.41: Effect of the parameter *burnin* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

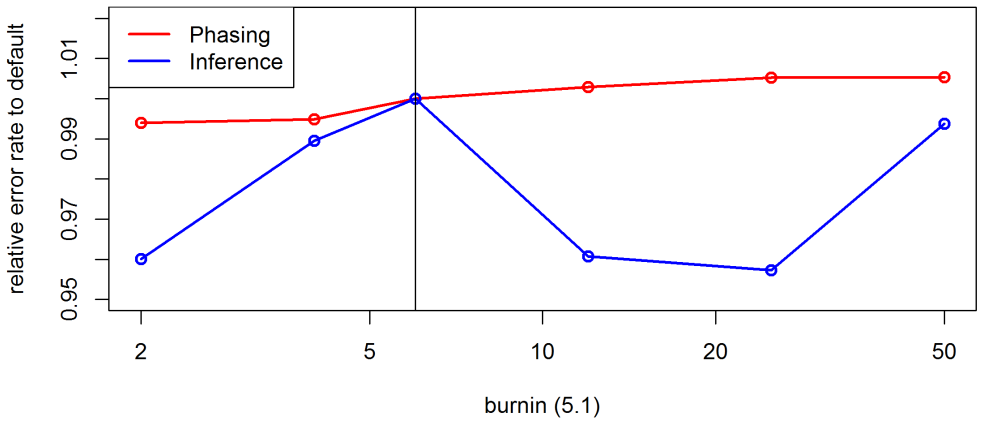


Figure 3.42: Effect of the parameter *burnin* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

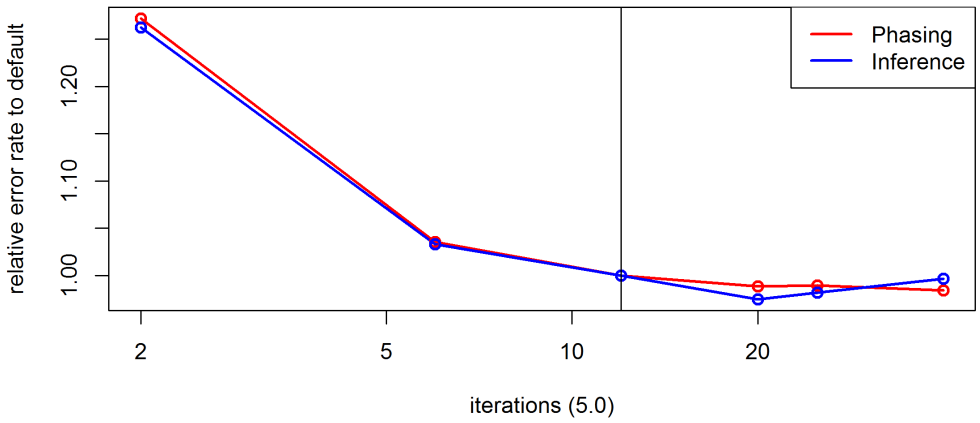


Figure 3.43: Effect of the parameter *iterations* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

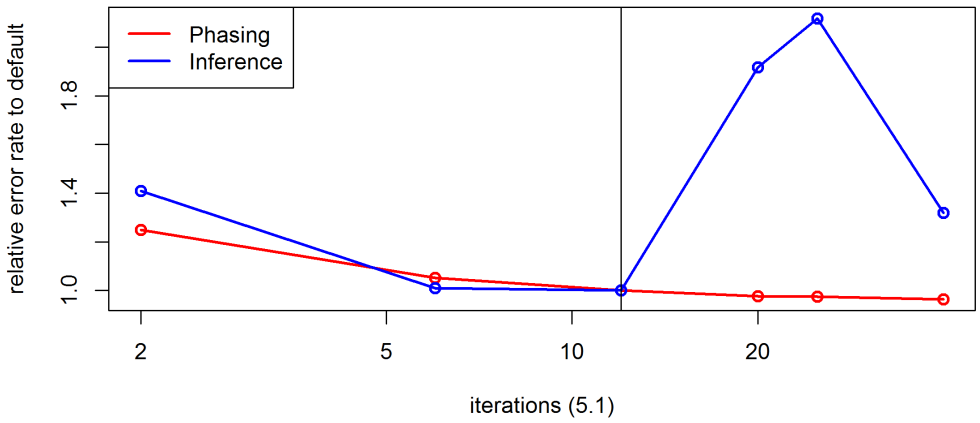


Figure 3.44: Effect of the parameter *iterations* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

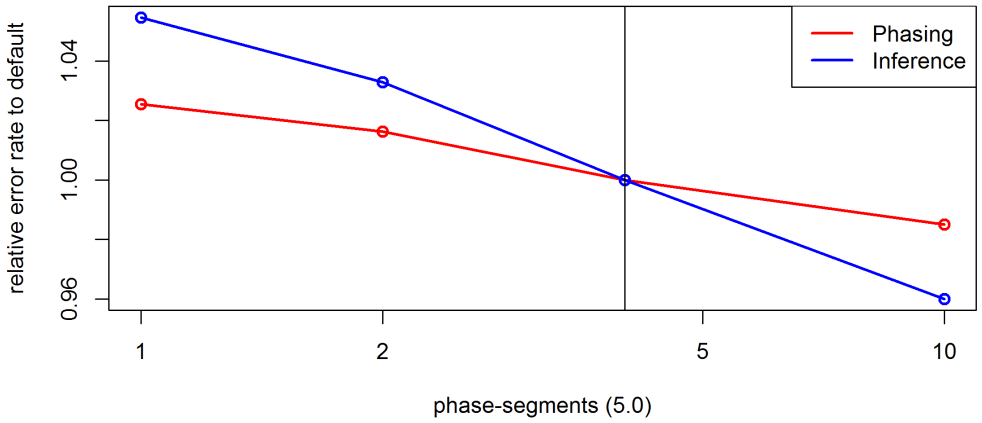


Figure 3.45: Effect of the parameter *phase-segment* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

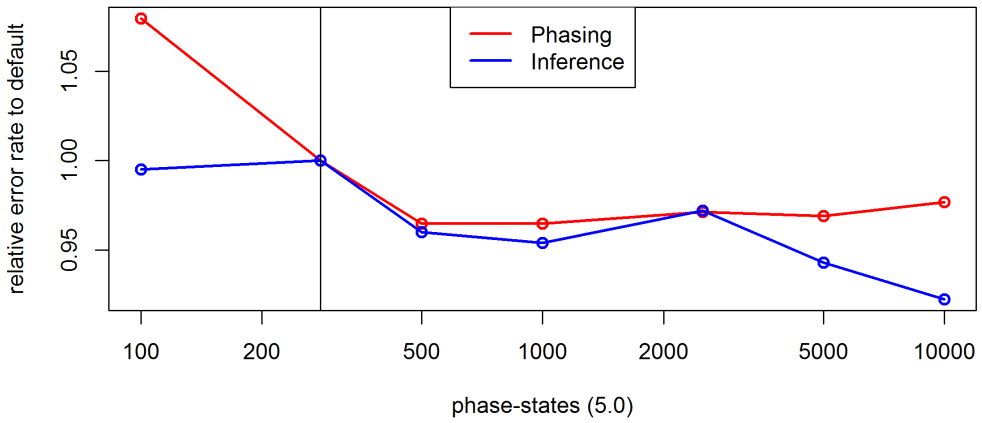


Figure 3.46: Effect of the parameter *phase-states* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line.

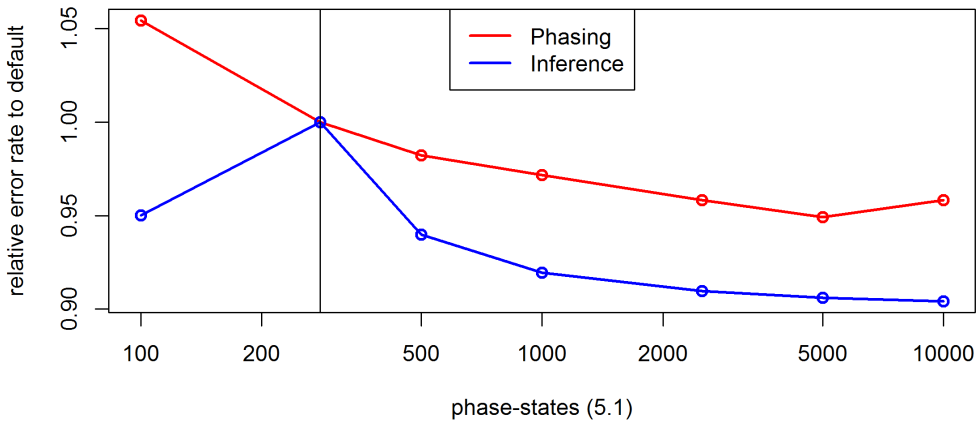


Figure 3.47: Effect of the parameter *phase-states* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line.

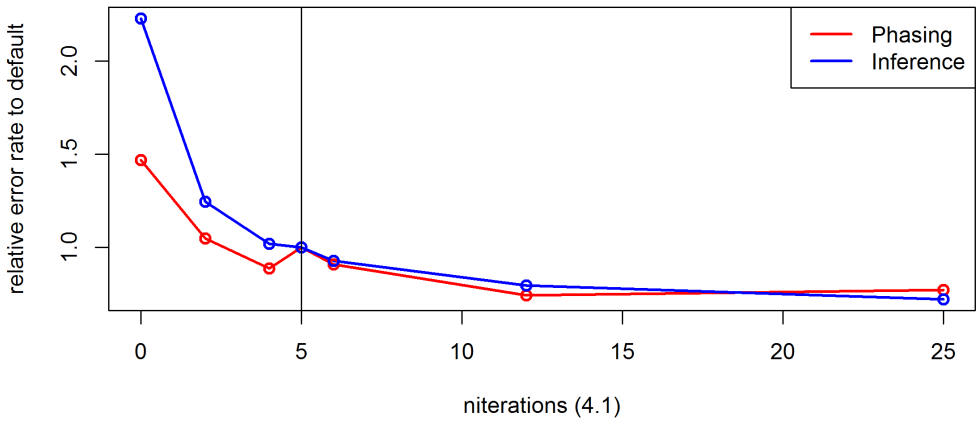


Figure 3.48: Effect of the parameter *niterations* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line.

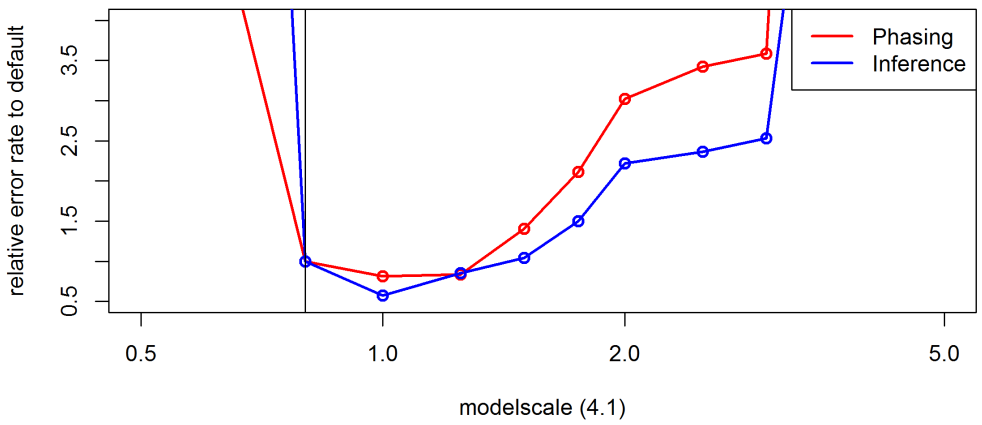


Figure 3.49: Effect of the parameter *modelscale* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line.

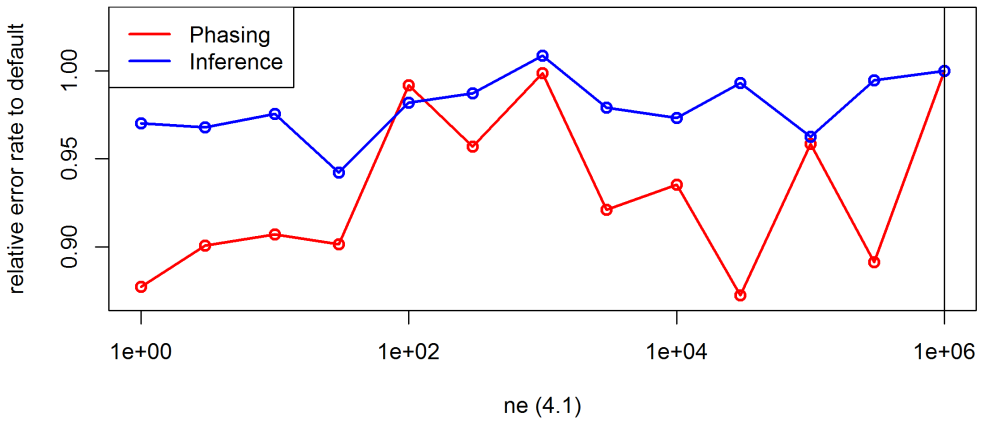


Figure 3.50: Effect of the parameter *ne* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line.

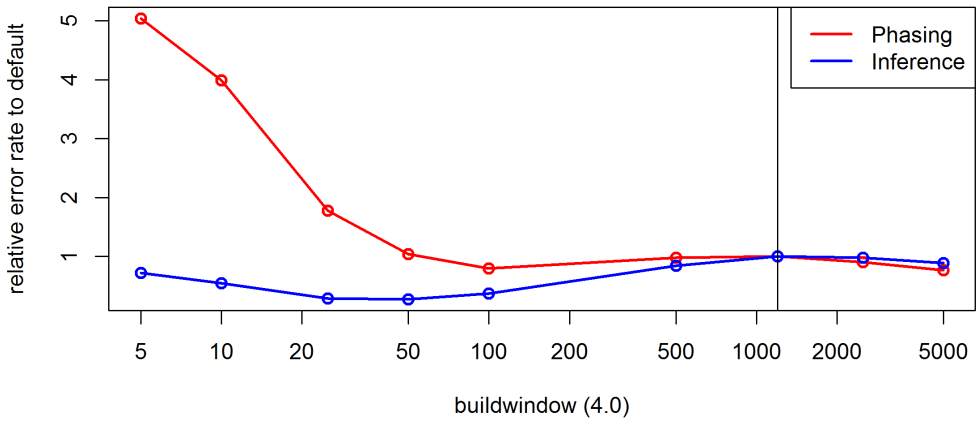


Figure 3.51: Effect of the parameter *buildwindow* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

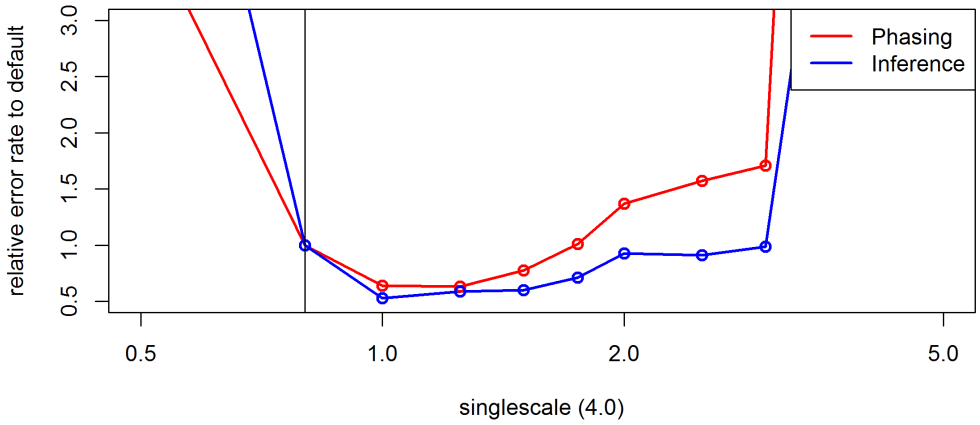


Figure 3.52: Effect of the parameter *singlescale* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

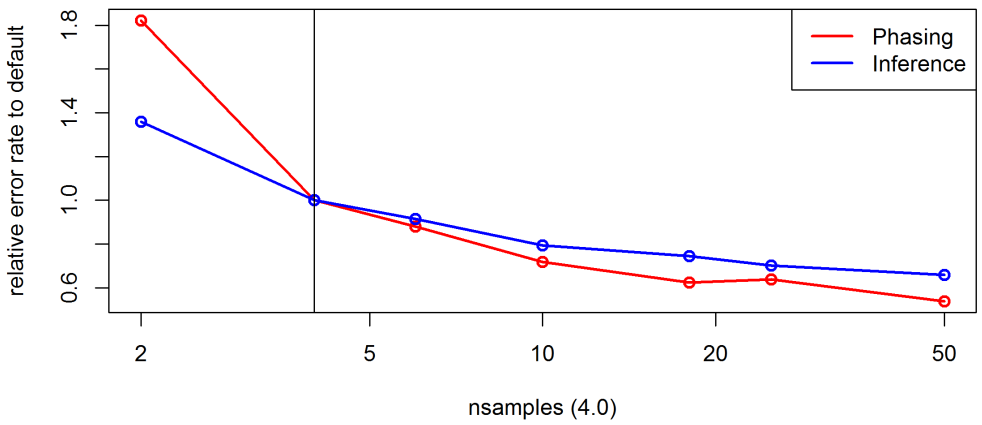


Figure 3.53: Effect of the parameter *nsamples* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

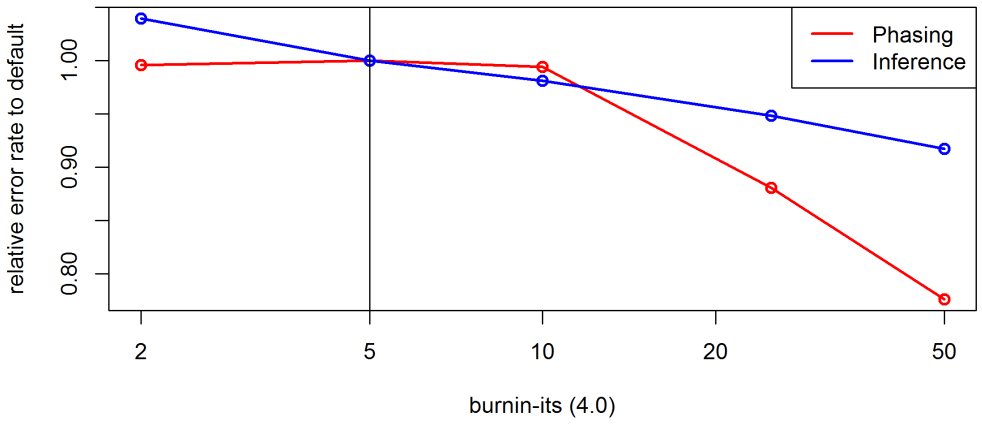


Figure 3.54: Effect of the parameter *burnin-its* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

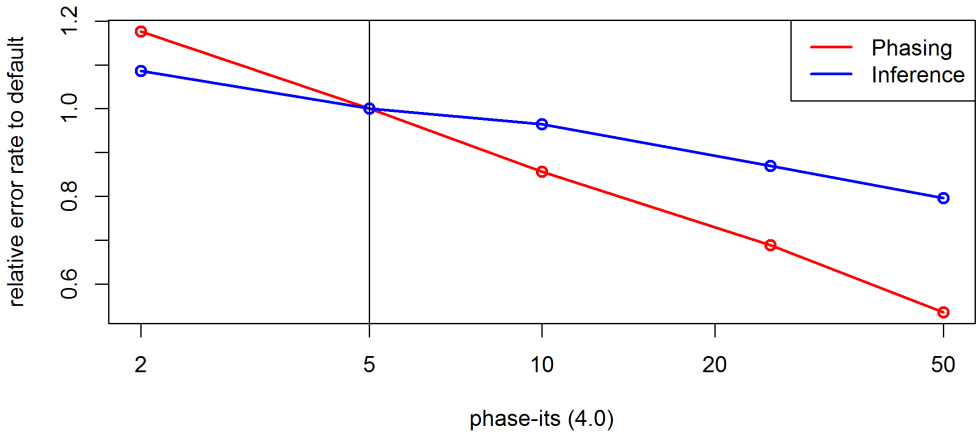


Figure 3.55: Effect of the parameter *phase-its* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line.

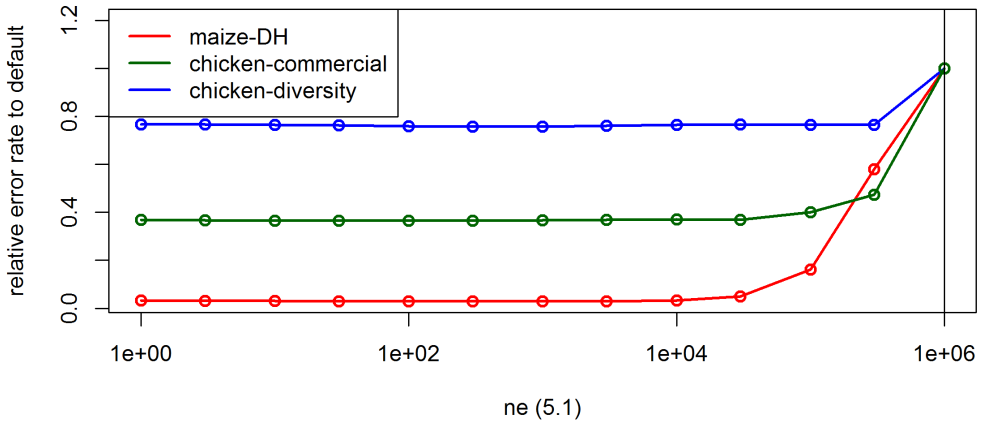


Figure 3.56: Effect of the parameter *ne* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

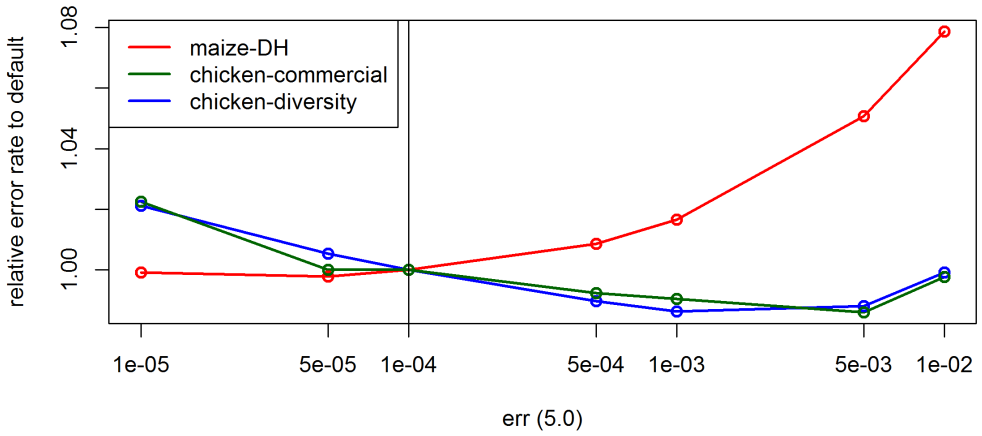


Figure 3.57: Effect of the parameter *err* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

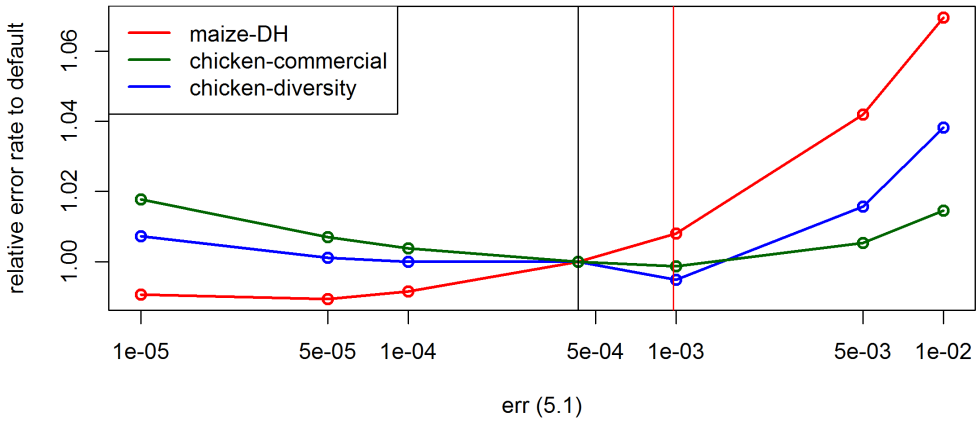


Figure 3.58: Effect of the parameter *err* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical lines (black for chicken, red for maize).

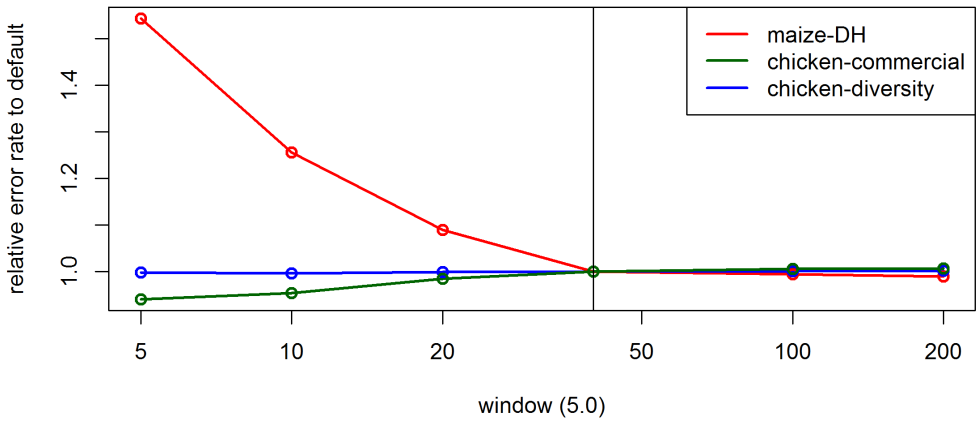


Figure 3.59: Effect of the parameter *window* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

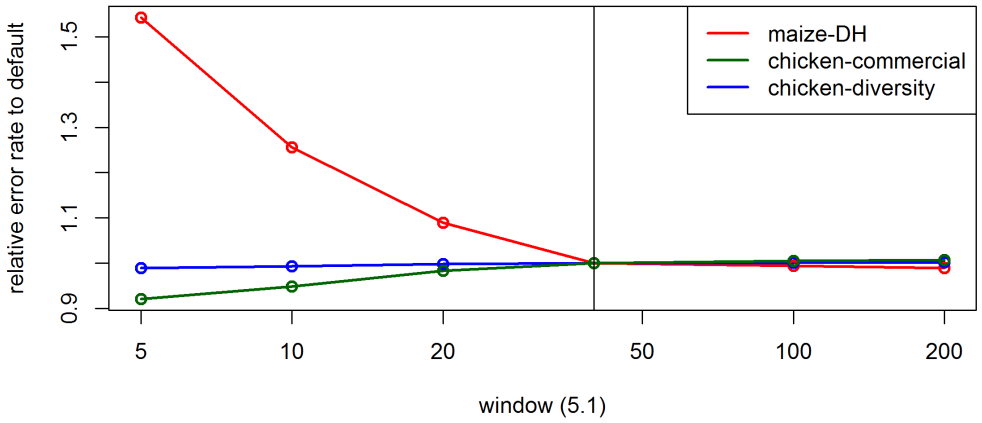


Figure 3.60: Effect of the parameter *window* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

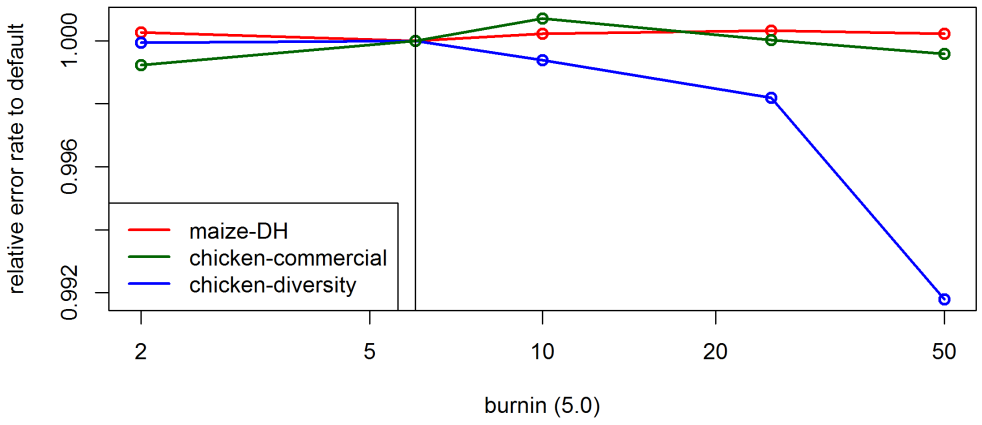


Figure 3.61: Effect of the parameter *burnin* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

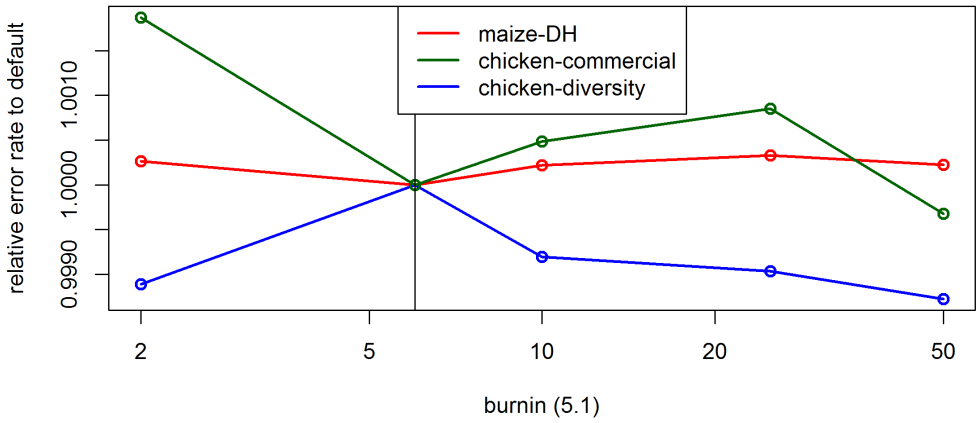


Figure 3.62: Effect of the parameter *burnin* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

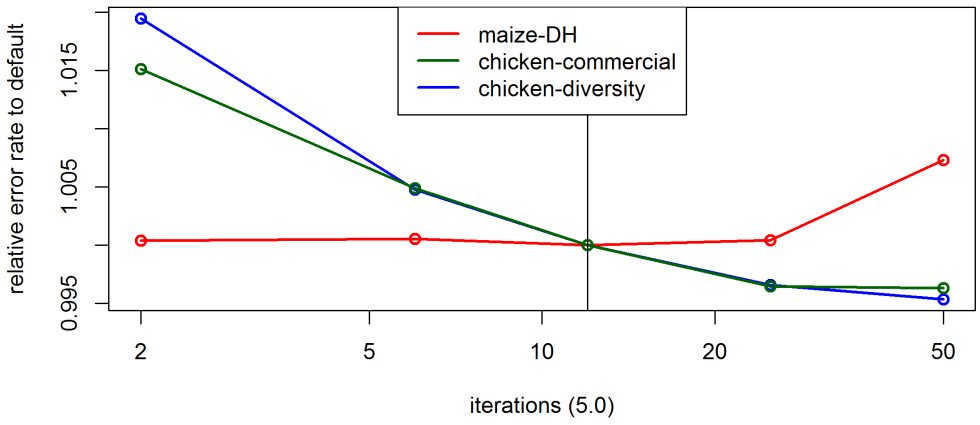


Figure 3.63: Effect of the parameter *iterations* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

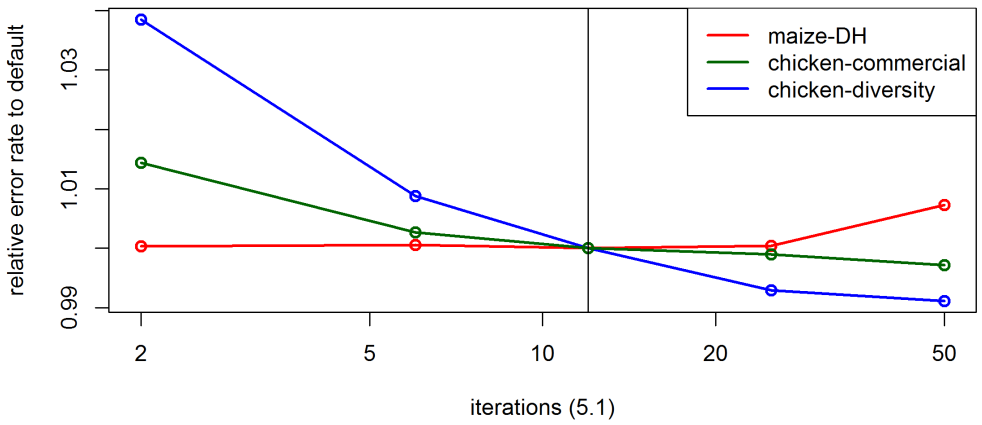


Figure 3.64: Effect of the parameter *iterations* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

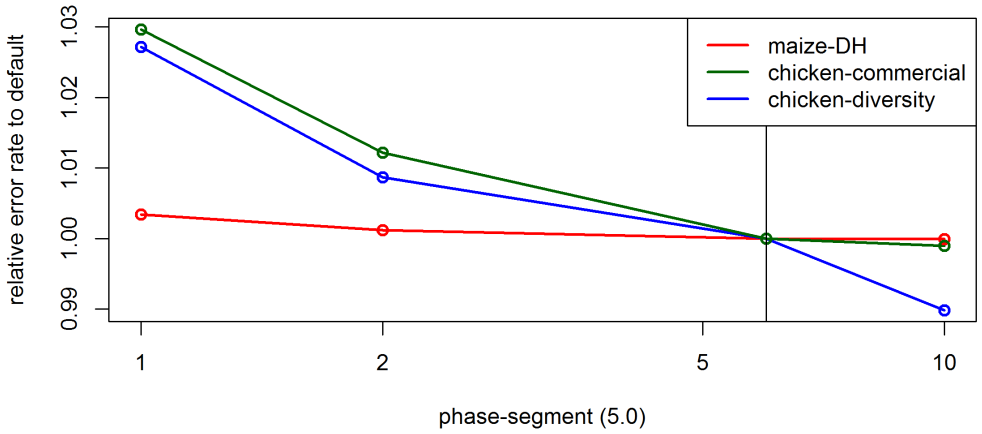


Figure 3.65: Effect of the parameter *phase-segment* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

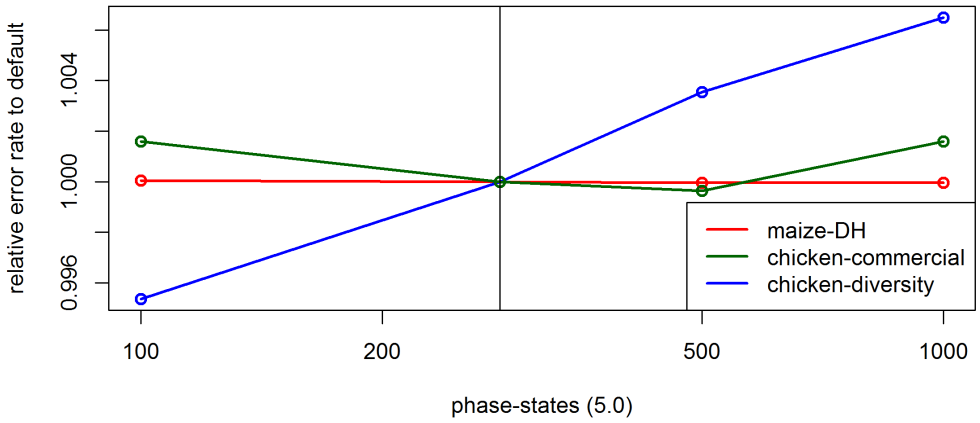


Figure 3.66: Effect of the parameter *phase-states* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

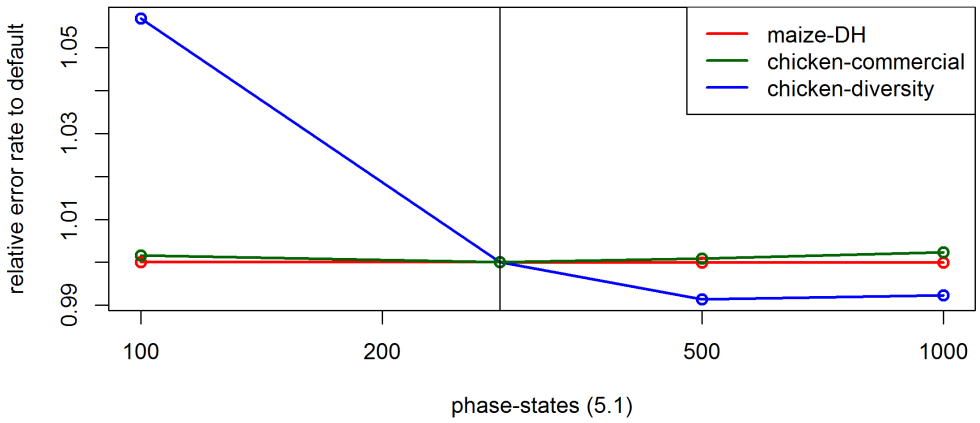


Figure 3.67: Effect of the parameter *phase-states* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

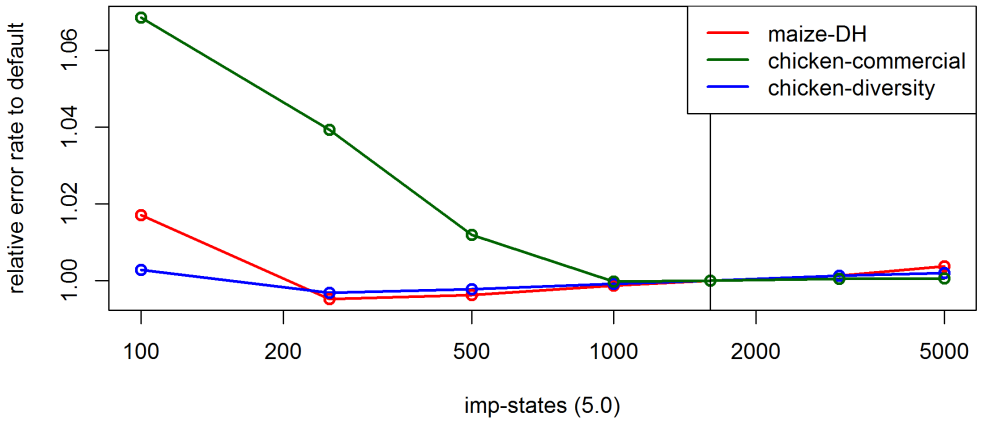


Figure 3.68: Effect of the parameter *imp-states* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

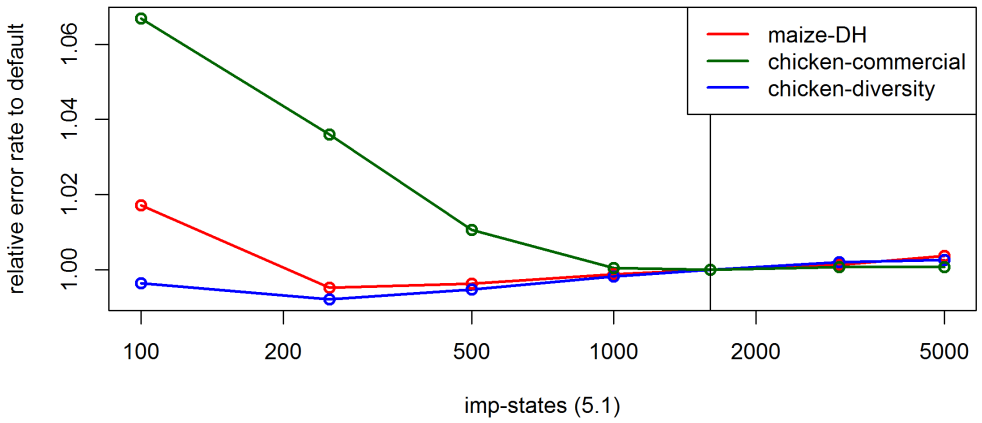


Figure 3.69: Effect of the parameter *imp-states* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

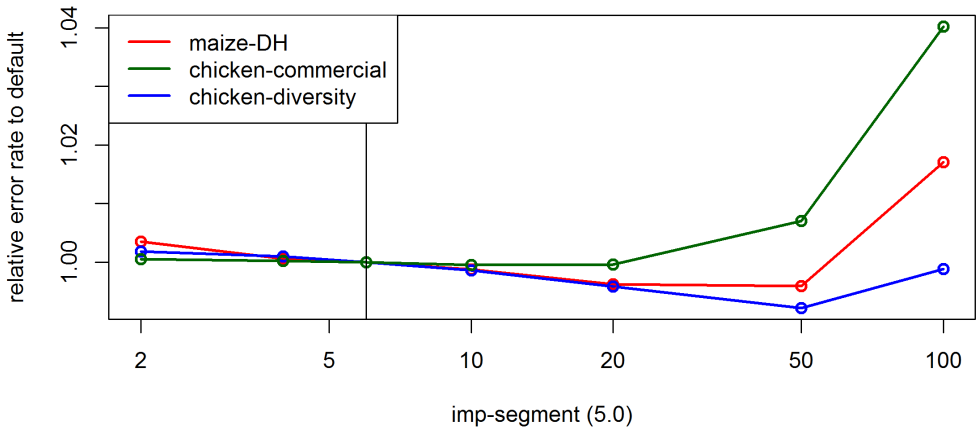


Figure 3.70: Effect of the parameter *imp-segment* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

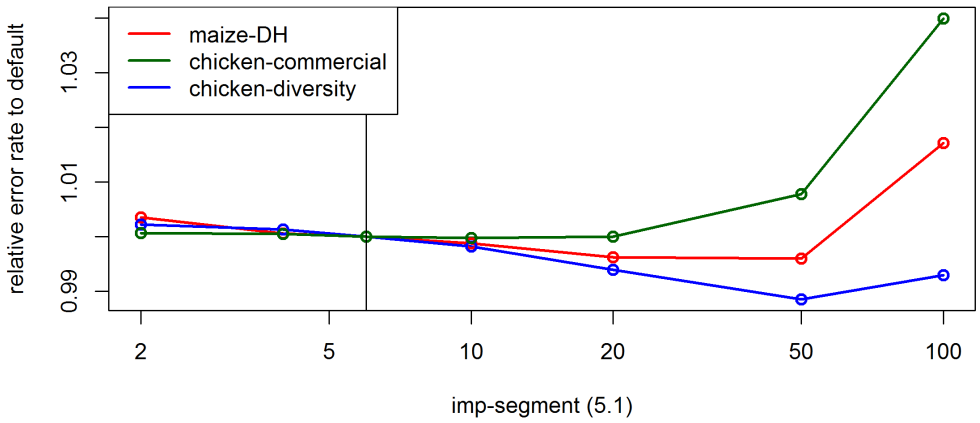


Figure 3.71: Effect of the parameter *imp-segment* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

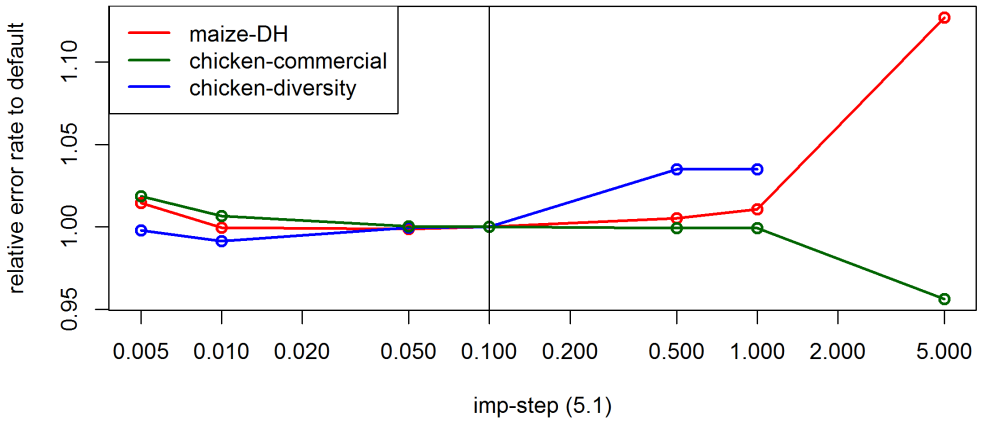


Figure 3.72: Effect of the parameter *imp-step* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

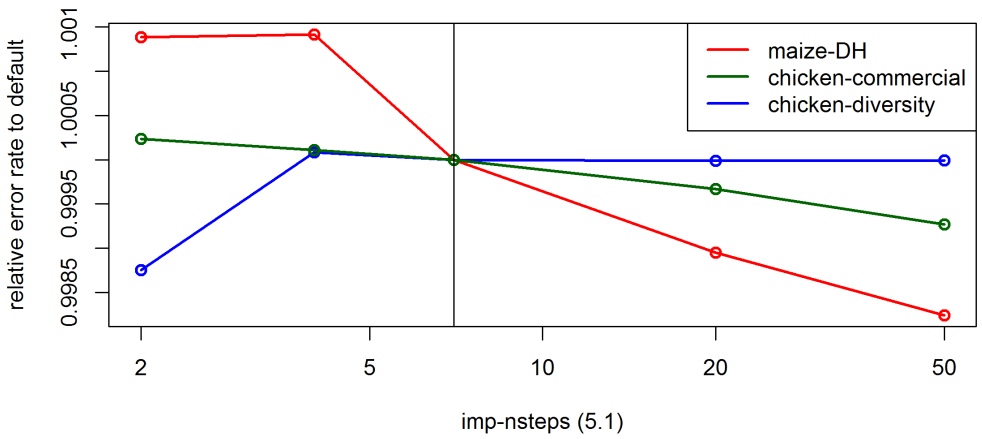


Figure 3.73: Effect of the parameter *imp-nsteps* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

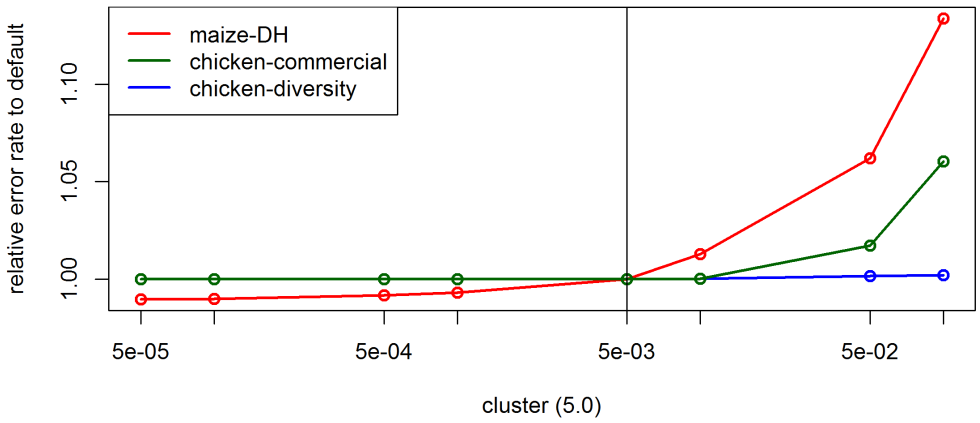


Figure 3.74: Effect of the parameter *cluster* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line.

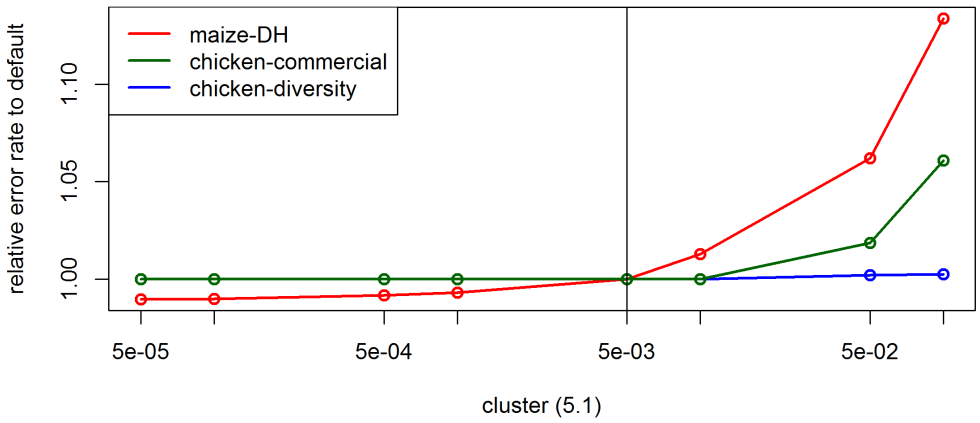


Figure 3.75: Effect of the parameter *cluster* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line.

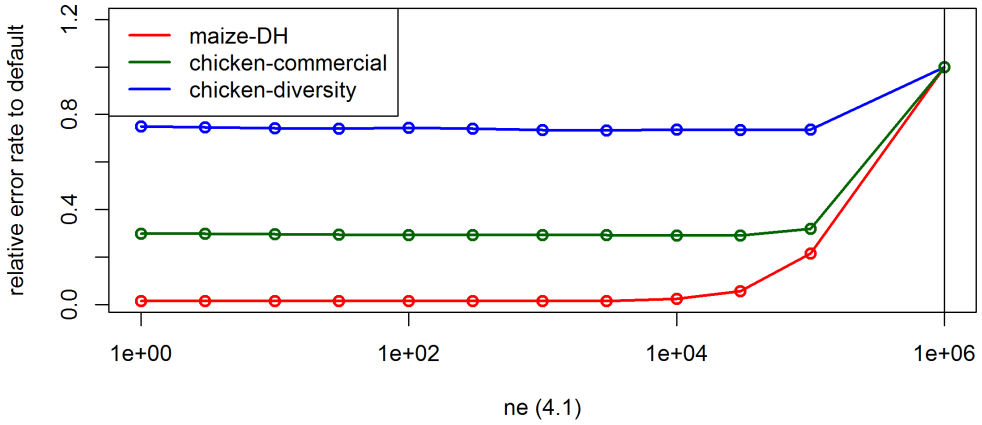


Figure 3.76: Effect of the parameter *ne* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.1. Default settings are indicated by the vertical line.

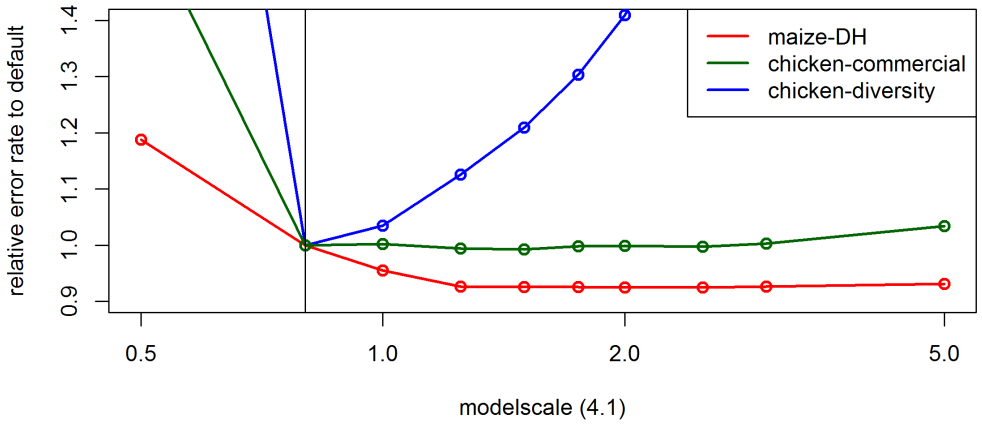


Figure 3.77: Effect of the parameter *modelscale* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.1. Default settings are indicated by the vertical line.

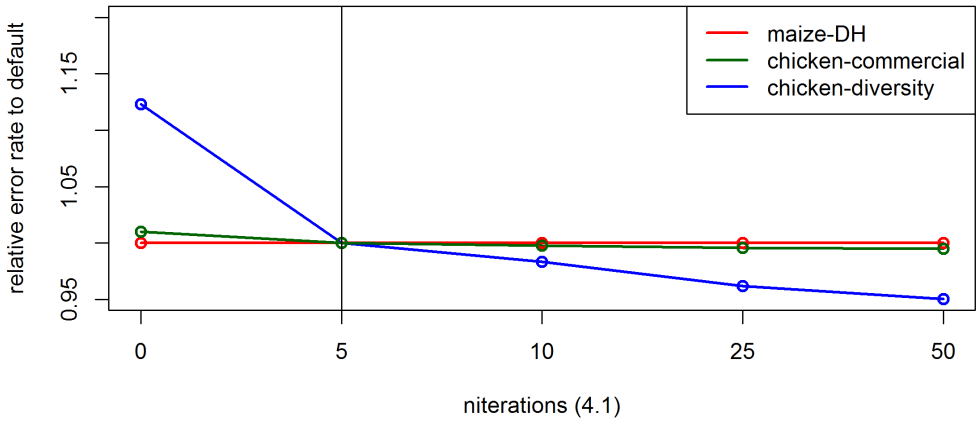


Figure 3.78: Effect of the parameter *niterations* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.1. Default settings are indicated by the vertical line.

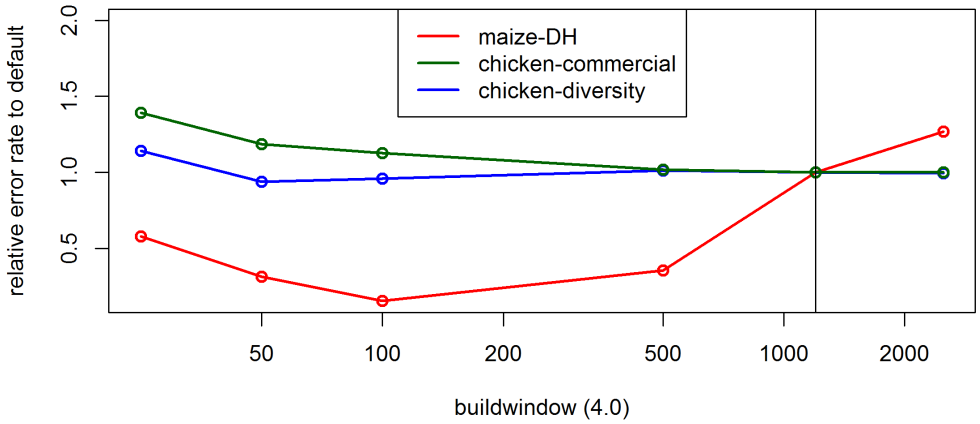


Figure 3.79: Effect of the parameter *buildwindow* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line.

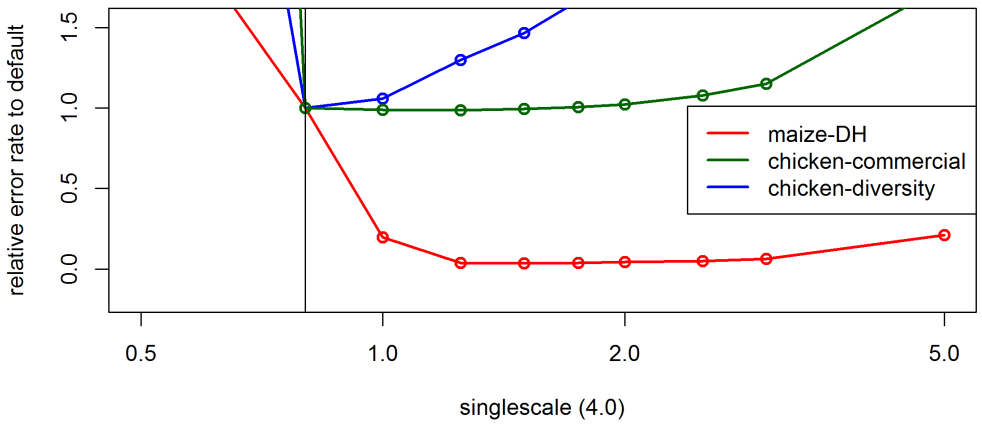


Figure 3.80: Effect of the parameter *singlescale* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line.

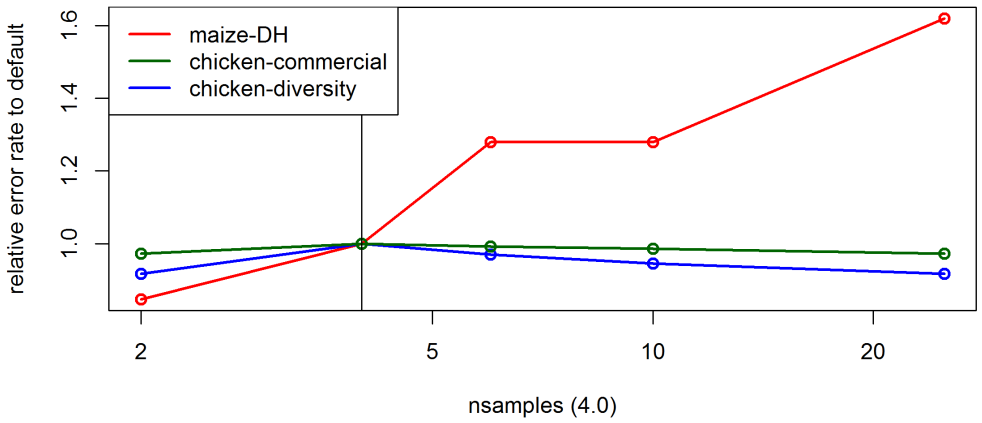


Figure 3.81: Effect of the parameter *nsamples* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line.

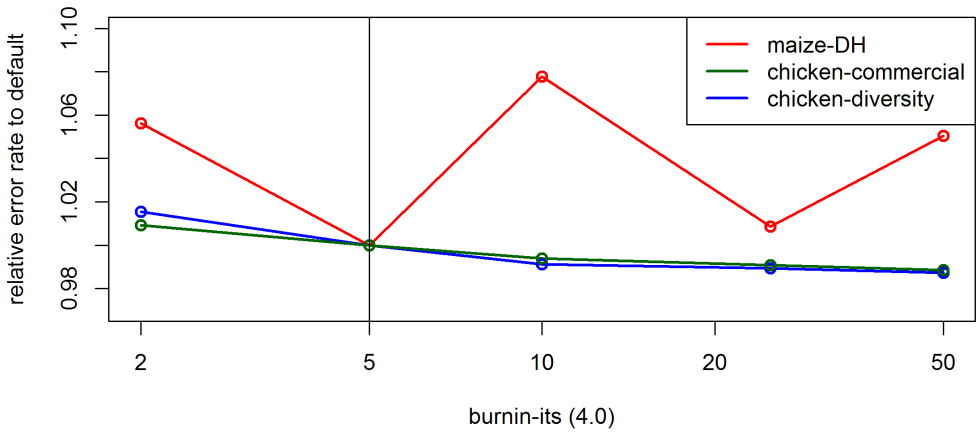


Figure 3.82: Effect of the parameter *burnin-its* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line.

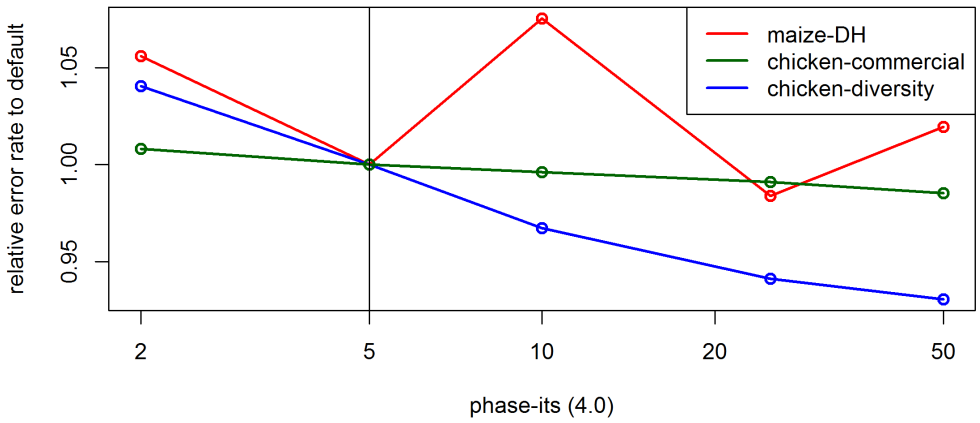


Figure 3.83: Effect of the parameter *phase-its* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line.

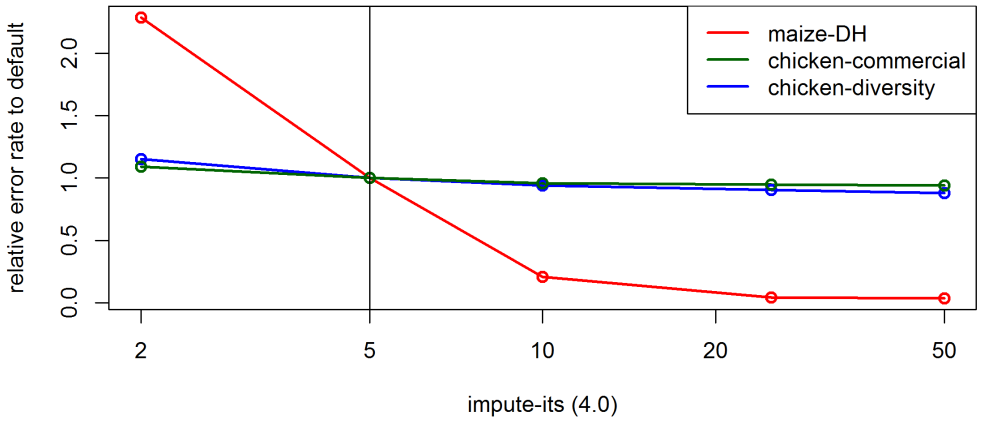


Figure 3.84: Effect of the parameter *impute-its* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line.

4 MoBPS - Modular Breeding Program Simulator

"An idea not coupled with action will never get any bigger than the brain cell it occupied."
Arnold Glasow

This chapter contains the manuscript "MoBPS - Modular Breeding Program Simulator" that is currently in review at the journal *G3: Genes, Genome, Genetics*. For reasons of uniformity in this thesis, the journal style is not used in this chapter. The manuscript is an application note for the R-package MoBPS (R Core Team, 2017; Pook et al., 2018). For an overview of exemplary simulations that can be performed the interested reader is referred to Chapter 5.3 and 5.4. The current version of the R-package and an in-depth user manual is available at `https://github.com/tpook92/MoBPS` and Supplementary B.

MoBPS: R-package

This work is a joined work of Torsten Pook[1,2], Martin Schlather[2,3] and Henner Simianer[1,2].

1: Department of Animal Sciences, University of Goettingen, Goettingen, 37075, Germany
2: Center for Integrated Breeding Research, University of Goettingen, Goettingen, 37075, Germany
3: Stochastics and Its Applications Group, University of Mannheim, Mannheim, 68131, Germany

Author contributions by TP

TP lead the development of the methodology, performed most of the analysis, wrote the associated R-package (C parts written by MS), wrote the initial manuscript and led the revision of the manuscript.

4.1 Abstract

The R-package MoBPS provides a computationally efficient and flexible framework to simulate complex breeding programs and compare their economic and genetic impact. Simulations are performed on the base of individuals and haplotypes are calculated on-the-fly by only saving founder haplotypes, points of recombination and mutations. MoBPS utilizes a highly efficient implementation with bit-wise storage of data and matrix multiplications from the associated R-package miraculix allowing to handle large scale populations. The modular structure of MoBPS allows to combine rather coarse simulations, as needed to generate founder populations, with a very detailed modeling of todays' complex breeding programs, making use of all available biotechnologies. MoBPS provides pre-implemented functions for common breeding practices such as optimum genetic contributions and single-step GBLUP but also allows the user to replace certain steps with personalized and/or self-written solutions.

4.2 Introduction

Breeding programs aim at improving the genetic properties of livestock and crop populations w.r.t. productivity, fitness and adaptation. Progress towards the target is limited by the available resources, but also negative effects, such as inbreeding depression or health issues, have to be avoided or at least controlled. Hence, the allocation of resources in a breeding program is a complex optimization problem. Additionally, population history, such as fluctuating population sizes and selection pressures, has an impact on the current genomic architecture and thus the potential for future improvement.
Over the years a variety of simulation tools have been developed to assist breeders to evaluate and optimize their breeding programs. A general problem of simulation studies is that the underlying genomic processes are highly complex and have to be simplified for modeling. In addition, users often have rather different objectives in mind when setting up their simulation studies. Since tools often do not provide the necessary flexibility to execute the specific breeding actions and/or it is not possible to export all necessary outputs, this commonly leads to the use of self-developed solutions that tend to be more error-prone, less sophisticated and computationally inefficient. The functionality of existing software ranges from cohort based deterministic simulation that relies on expected gains like ZPLAN+ (Täubert et al., 2010) to applications on the base of the stochastic simulation of single individuals such as QMSim (Sargolzaei and Schenkel, 2009) and AlphaSim (Faux et al., 2016). The functionality of each of these tools highly depends on the intended use. ZPLAN+ (Täubert et al., 2010) focuses on the economic impact from a macro-perspective. Since analytic formulas for cohorts are required, it has limitations when simulating complex mating schemes or when focusing on other quantities than genetic or economic gain. QMSim (Sargolzaei and Schenkel, 2009) is able to simulate each

individual meiosis but is limited in the options for the design of the breeding program itself. As QMSim is mostly designed for population genetic studies, a typical application of the tool is the generation of a historical population, often followed by self-developed solution in later steps. On the contrary, AlphaSim (Faux et al., 2016) provides a lot of flexibility in term of the design of the breeding program, especially for plant breeding and when the number of cohorts in the breeding program is small. However, AlphaSim lacks the efficiency and flexibility to simulate complex and large scale populations.
Our goal was to develop a tool that combines the simulation of a historical population and the evaluation of a subsequent complex breeding program in a computationally efficient way. The Modular Breeding Program Simulator (MoBPS) is not only flexible in terms of parameters and design of breeding programs, but also allows the user to replace standard procedures of the package with own ones.

4.3 Methods

Simulations in MoBPS are ultimately based on the simulation of single individuals. In principle, this allows the user to control each singular mating and modify recombination or mutation rates for the respective meiosis. Breeding programs are constructed in a modular form as a combination of cohorts (Hill, 1974), representing a group of contemporary individuals with similar characteristics, and transformations, which link one or several parent cohorts to a child cohort. Examples for such transformations are aging, selection, or reproduction, and each transformation reflects a set of rules how the characteristics of the parent cohort(s) are transformed into the characteristics of the child cohort. Cohorts and transformations are defined in a generic way and are parametrized, so that any breeding program of arbitrary complexity can be modeled as a suitable sequence of cohorts and transformations.
All data for a population is stored in a list that contains general and individual information. The general part provides information on the underlying genetics like the physical position of each marker, allelic variants or structure of the underlying genetic traits. The individual part contains information that is specific to the individual. Haplotypes are stored for founder individuals only. For all other individuals only points of recombination and mutation and their genetic origins are stored and haplotypes are derived on-the-fly. Therefore, the required memory is minimized and only increases slightly with increasing marker density. When thousands of generations are simulated it is advisable to classify additional generations as new founders to reduce the number of recombinations and mutations to be stored in subsequent generations.
Simulation of multiple correlated traits with and without underlying QTL is supported. Classical additive, dominant and epistatic or pleiotropic QTL can be defined and any effect structure of multiple interacting loci is supported. Each locus has to be assigned with a position in Morgan and different recombination rates for subgroups (e.g. males/females) are supported. Information on the number of markers can be manually entered or imported via a database (Ensemble, (Zerbino et al., 2017)),

a map-file (Purcell et al., 2007) or a vcf-file (Danecek et al., 2011). For common species, exemplary map files are provided in the associated package MoBPSmaps (Pook, 2019b). Genotype data for a base population can be imported via PLINK (Purcell et al., 2007) and/or vcf-format (Danecek et al., 2011), sampled internally or generated by executing prior simulation in MoBPS and/or other tools (Chen et al., 2009; Sargolzaei and Schenkel, 2009) to generate the required population structure. All breeding actions performed in the simulation can be tracked and assigned with costs to derive the expenses of the program. Different breeding programs can be compared in terms of their economic revenue or other target functions (e.g. development of the inbreeding rate) one is interested in.
Common methods for selection such as optimal genetic contributions (Meuwissen, 1997) are implemented and a variety of different packages for breeding value estimation can be switched on. This includes BGLR (Pérez and de los Campos, 2014), sommer (Covarrubias-Pazaran, 2016) and rrBLUP (Endelman, 2011), as well as an efficient implementation for solving the mixed model (Henderson, 1975) in the traditional GBLUP model (Meuwissen et al., 2001; VanRaden, 2008) that is assuming known heritability and is using the R-package RandomFieldsUtils (Schlather et al., 2019b) for the matrix inversion. Inputs for these packages such as the different pedigree and genomic relationship matrices (VanRaden, 2008; Legarra et al., 2014; Martini et al., 2017) can be derived via highly efficient and fully-parallelized bit-wise matrix multiplications (R-package miraculix (Schlather et al., 2019a)). Non of the mentioned packages, however, is required to execute simulations in MoBPS. In particular all functionality of the MoBPS R-package is still available when miraculix is not installed, with the downside of higher computing times and memory demands.
The simulations in MoBPS are based on two main functions: *creating.diploid()* and *breeding.diploid()*. Here, *creating.diploid()* initializes the base-line population and *breeding.diploid()* performs breeding actions on an existing population list. As a simple example consider the following script:

```
library(MoBPS)
pop <- creating.diploid(nsnp = 10000, nindi = 50, chr.nr = 5, chromosome.length = 2,
          n.additive = 25, n.dominant = 5, name.cohort = "Founder")
pop <- breeding.diploid(pop, heritability = 0.5, new.bv.observation = "all")
pop <- breeding.diploid(pop, bve = TRUE)
pop <- breeding.diploid(pop, breeding.size = 50, selection.size = c(5,10),
     selection.m = "function", selection.m.cohorts = "Founder_M",
     selection.f.cohorts = "Founder_F", name.cohort = "Offspring")
```

Via this code, we first generate a base population containing 50 individuals with 10'000 markers. The underlying genome consists of 5 chromosomes with a length of 2 Morgan each and equidistant markers. Furthermore, we generated a single trait that is impacted by 25 purely additive QTLs and 5 dominant QTLs.
In the next step, we initialize a breeding action to generate phenotypes for all individuals in the population with an assumed heritability of 0.5. Next, a breeding value estimation is performed. Since no cohorts are selected, the last (and only) generation of the population list will be considered for the breeding value estimation. Lastly, we generate 50 offspring by randomly mating the top 5 male and top 10 female individuals. In principle, all three breeding actions performed via *breeding.diploid()* could have also been executed in a joint step. For a full list of all possible breeding actions and available parameters we refer to our user manual

(available at `https://github.com/tpook92/MoBPS`).
For a quick overview of the simulated population, the function *summary()* can be used:

```
> summary(pop)
Population size:
Total: 100 Individuals
Of which 50 are male and 50 are female.
There are 2 generations
and 4 unique cohorts.

Genome Info:
There are 5 unique chromosomes.
In total there are 10000 SNPs.
The genome has a total length of 10 Morgan.
No physical positions are stored.

Trait Info:
There is 1 modelled trait.
The trait has underlying QTL
The trait is named: Trait 1
```

A variety of functions is provided to export required information such as the phenotypes (*get.pheno()*), the genotypes (*get.geno()*) and the pedigree (*get.pedigree()*) for selected individuals from the population list. These functions are thoroughly described in chapter 9 of the user manual (available at `https://github.com/tpook92/MoBPS`). Furthermore, functions to derive rates of inbreeding (*kinship.emp()*), development of breeding values (*bv.development()*) or changes in allele frequency over time (*analyze.population()*) are provided to further analyze the resulting population list.

4.4 Results and Discussion

The package MoBPS is completely written in R (R Core Team, 2017) so that all functionalities for genetic applications are platform independent. The R-packages miraculix (Schlather et al., 2019a) can be activated in MoBPS and leads to more efficient data storage and shorter simulation times. In particular vector multiplications with genetics data (0,1,2) are performed via bitwise operations on a whole register (128/256 bit) using SSE2/AVX2. Computing times are similar to the ones in PLINK (Purcell et al., 2007) with one fourth of the memory usage.
Even though basically all information regarding each individual is stored, the required memory in MoBPS is still relatively low as a highly efficient storage structure is used. Haplotypes of founders and details on the origin of the segment between points of recombination are stored bitwise. E.g. the simulation of 20 generations with 50'000 cows with 50'000 markers and breeding value estimation via GBLUP takes 26.2 hours using 24 cores on a server cluster with Intel E5-2650 (2X12 core 2.2GHz) processors. At peak, 65 GB of memory was used. The main share of this was required for the storage of the genomic relationship matrix whereas the resulting population list, containing more than a million individuals, only had a size of about 0.44 GB. The biggest proportion of the computing time is used for breeding value estimation (25.3 hours, 96.4%). The generation of new animals took 55 minutes

(3.5%, 304 animals per second using a single core). All other parts needed negligible computing time (132 seconds, 0.1%). Computing times for most parts (except breeding value estimation) increase linearly with the number of individuals. This highly efficient storage structure therefore also allows for the simulation of historical populations with thousands of generations and undergone population dynamics such as genetic bottlenecks, migration or mutational drift.
The flexible and efficient enviroment of MoBPS allows for the simulation of a variety of different and potential large-scale breeding programs. For exemplary scripts of more complex breeding programs we refer to the user manual. Exemplary simulations are given for the effect of gene editing in a cattle breeding program (Simianer et al., 2018), the simulation of a multi-parent advanced generation intercross in maize (Pook et al., 2019), an introgression scheme in chicken (Ha et al., 2017) and the generation of a base population with a hard sweep. A further advantage of MoBPS in comparison to other simulation tools is its flexible structure that allows the user to substitute single steps of the breeding program with a personalized and/or self-written solution. For this consider the following example to execute one owns breeding value estimation:

```
genos <- get.geno(pop, gen=1)
y <- get.pheno(pop, gen=1)
indi_names <- colnames(genos)

# Execute one owns function to perform the breeding value estimation
y_hat <- own.method.for.bve(genos, y)

# Enter BVEs in the population-list
pop <- insert.bve(pop, bves = cbind(indi_names, y_hat))
```

Even though a simulation study can never fully reflect reality and is relying on model assumptions, the use of a simulation study comes with major benefits and still allows the user to draw important conclusions. In contrast to reality the underlying truth in a simulation study is known, and therefore new methods can be thoroughly evaluated and compared to existing ones. Furthermore, the effects of particular breeding actions on a variety of output dimension can be assessed and compared. This in turn can be used to derive an ideal resource allocation and optimize potentially highly complex breeding scenarios in a setting that can be evaluated multiple times and without constrains both in terms of money and time.

4.5 Web resources

An executable version of MoBPS and the associated R-packages miraculix (Schlather et al., 2019a), RandomFieldsUtils (Schlather et al., 2019b) and MoBPSmaps (Pook, 2019b) for Windows and Linux are freely available at `https://github.com/tpook92/MoBPS`. This directory also contains an comprehensive user manual explaining the functionality of all input parameters and utility functions in MoBPS. A frozen version of the R-packages MoBPS (v1.4.15), MoBPSmaps (v0.1.7), miraculix (v0.9.7), RandomFieldsUtils (v0.5.9), and our user manual at submission are also provided there. The MoBPS R-package can be directly installed within your R session via following commands:

```
1 install.packages("devtools")
2 devtools::install_github("tpook92/MoBPS", subdir = "pkg")
```

Acknowledgments

This package was developed in the context of the European Union's Horizon 2020 Research and Innovation Program under grant agreement n°677353 IMAGE.

5 Discussion

"Imagination is more important than knowledge. Knowledge is limited.
Imagination encircles the world."
Albert Einstein

5.1 Influence of imputation quality on haplotyping methods

The influence of imputation quality has to be assessed with respect to the application the resulting data are used with. Since missing data usually leads to problems in the computations, inference is needed for basically all applications. In contrast, imputation of ungenotyped markers (UM imputation) and phasing can sometimes be neglected. Whereas UM imputation has been shown to be a powerful tool for GWAS (Klein et al., 2005; Yan et al., 2017), the benefit for genomic prediction is often not given and obtained results can sometimes even be improved by LD pruning (Barrett et al., 2005) to reduce the dimensionality of the dataset (Chapter 5.2). Overall, the higher number of markers has to be weighted against a potential higher proportion of errors via UM imputation. The accuracy of phasing has an impact on inference and UM imputation, but the phase itself is often not required for further analysis.

LD-based haplotyping approaches do not utilize the haplotype phase, therefore phasing errors can be neglected. Especially for pairs of markers of medium allele frequency the correlation, and thus LD (r^2), is robust against minor deviations. For this reason, inference errors should not heavily influence the results of LD-based haplotyping. Depending on the algorithm used for the derivation of blocks, the marker density and thus UM imputation can have a major impact on the block structure. Using a method like the one proposed in Gabriel et al. (2002) that is implemented in HaploView (Barrett et al., 2005), blocks are generated by merging all markers with a certain pairwise minimum of D' (Lewontin, 1964; Falconer and Mackay, 1996). Since imputation leads to a higher number of potential pairs for evaluation, it is commonly observed that the physical length of blocks decreases with increasing marker density. The interested reader is referred to Kim and Yoo

(2016) for a detailed analysis of the influence of the marker density on the resulting haplotype blocks.

The detection of IBD is heavily influenced by the phasing accuracy as haplotypes are required in the detection process. When allowing for minor deviations of the allelic sequences, inference and UM imputation should both only have a minor impact on the detection procedure. Especially for relatively short IBD segments, a high marker density can lead to more accurate detection of the segment boundaries. It should be noted that regions with high inference error tend to be prone to phasing errors and *vice versa*. Depending on the method used, IBD detection, phasing, inference and UM imputation can be heavily connected (Browning and Browning, 2011, 2013).

Since HaploBlocker is rather robust to minor deviations, both inference and UM imputation error only have a minor impact on the resulting haplotype blocks. As shown in Chapter 2, the structure of the haplotype library is basically independent of the marker density. The obtainable phasing accuracy in most livestock and crop datasets should be sufficient to detect long-range haplotype blocks. Besides the already low phasing error rates (Chapter 3), the actually occurring phasing errors tend to be clustered locally. Most phasing errors occur in rare variants that would not be part of a haplotype block with a high number of haplotypes in it or are even ignored because of the robustness of HaploBlocker. The use of BEAGLE for the imputation procedure can further enhance the identification of haplotype blocks since the overall structure of the haplotype cluster (Browning, 2006) used in BEAGLE (Browning et al., 2018) and the window cluster used in HaploBlocker (Chapter 2) are structurally similar. After deriving haplotype blocks, one could even consider performing an additional screening step to determine which haplotype blocks are potentially present in an individual based on both of its haplotypes.

In case of livestock and crop datasets with less related individuals or human datasets, phasing can become a substantial problem for the application of HaploBlocker. For datasets of high overall quality and sample size like those of the 1000 Genomes Project Consortium (2015), application is possible, but overall phasing accuracy for datasets of lower quality is potentially not sufficient and needs to be checked. It should be noted that ultra-long read sequencing techniques like nanopore sequencing (Branton et al., 2010; Jain et al., 2018) may help to solve these phasing problems in the future.

5.2 Genomic prediction using haplotype blocks

The haplotype library derived in HaploBlocker (Chapter 2) can be used to generate a block dataset. A block dataset contains dummy variables representing the presence/absence of a particular block (0 or 1) for each haplotype. In case of heterozygotes, the two haplotypes of an individual can be merged, resulting in a block dataset that encodes the number of times (0, 1 or 2) a haplotype block is present in each individual. Since the structure of this dataset is very similar to a SNP-dataset,

routine application like genomic prediction can be performed in a similar way. A potential benefit of using a block dataset is that local epistatic interactions, which cannot be captured by additive single marker effects, can be modelled. In addition, the chance of long segments like haplotype blocks to be the same by chance is substantially lower than for a single SNP, and by this potential noise can be reduced. This may be particularly relevant for datasets that contain less related individuals than those considered in this section.

The accuracy of genomic prediction for the European maize landraces Kemater Landmais Gelb (KE) and Petkuser Ferdinand Rot (PE) was evaluated using the traditional GBLUP model (Meuwissen et al., 2001) with the genomic relationship matrix according to VanRaden (2008). The DH-lines generated in the MAZE project were used here (Chapter 2) and prediction accuracies were derived for nine traits, including early vigor (EV) and plant height (PH) at different growing stages, days to silking (DtSILK), days to tassel (DtTAS) and root lodging (RL). Three different datasets were used to derive the genomic relationship matrix: The full SNP-dataset (501'124 SNPs), a pruned dataset (29'833 SNPs) and the block dataset containing 2'859 blocks for KE and 3'352 blocks for PE. For both landraces default settings of HaploBlocker were used to derive the haplotype libraries. All datasets were divided into a training set (80% of the lines) and a test set (20%). The sampling procedure was repeated 200 times.

For five of the nine traits, the block dataset resulted in the highest prediction accuracy for the test set but differences overall were small (Figure 5.1). The average prediction accuracy of the pruned dataset was higher than for the full SNP-panel for all traits. The full SNP-panel performed worst for seven of out the nine traits. Overall, results for PE were similar with slightly better performance of the full SNP-dataset (Supplementary C).

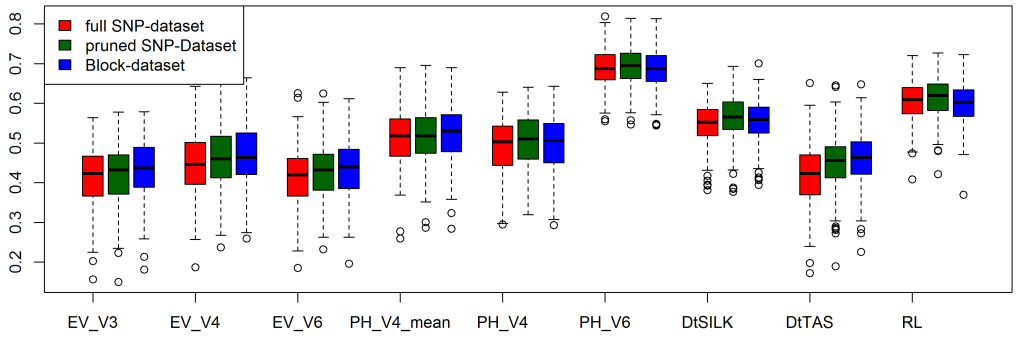


Figure 5.1: Accuracy of genomic prediction on the test set using different datasets to derive the genomic relationship matrix (VanRaden, 2008) for Kemater Landmais Gelb.

By reducing the haplotype library to a set of dummy variables, potentially valuable

information on the haplotype blocks themselves get lost. To incorporate information about the length of each block one could consider to modify the traditional formula to derive the genomic relationship matrix (VanRaden, 2008) to account for the length of each haplotype block:

$$G = \frac{ZW^sZ'}{2\sum_b {l_b}^s \cdot p_b\,(1-p_b)},$$

where Z is the block dataset, W is a diagonal matrix with entries l_b indicating the length of block b. The parameter s is a scaling factor for the relative weighting of the blocks depending on their length. Using $s = 0$ would result in the traditional genomic relationship matrix according to VanRaden (2008). For eight out of the nine traits a slightly increased weighting for longer haplotype blocks ($s \approx 0.5$) performed best for PE (Figure 5.2). An increased weighting of the block length had no positive effect in terms of prediction accuracy for KE (Supplementary C).

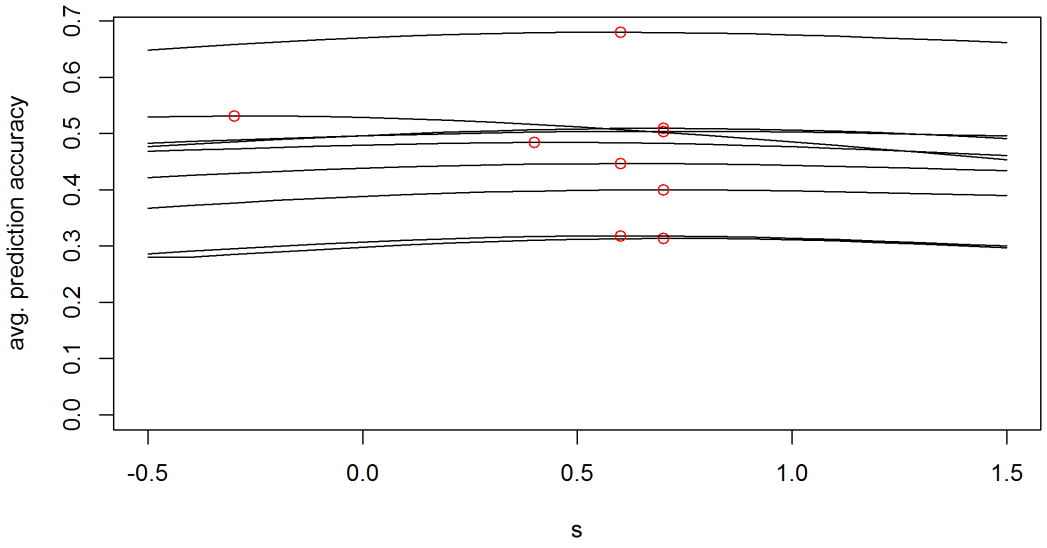


Figure 5.2: Accuracy of genomic prediction using different weightings s for the block length when deriving the genomic relationship matrix for Petkuser Ferdinand Rot. Each line is representing one trait and the red dot is indicating the maximum of the respective curve.

Overall, prediction accuracies when using haplotype blocks instead of a SNP-dataset were similar when performing within breed prediction. It can be concluded that either the share of epistatic effects for the traits considered is relatively low or that effects can at least be well approximated in the linear model. For the traits and landraces considered in this thesis, there was no gain in the use of high-density data over a smaller marker panel for genomic prediction. In addition, the much lower number of variables in the block dataset can enable a wide variety of new methods. In case of working with epistatic models (Vogjani et al., 2019) that include pairwise interactions between variables a model with 600'000 SNPs would lead to

almost 180 billion pairwise interactions to account for while 3'000 blocks result in about 4.5 million pairwise interactions. One might consider working with a model to combine the advantageous properties of both datasets. The most straight-forward implementation for this would be the use of a mixed model including two random effects (Henderson, 1975; Zhu, 1995) with variances according to the SNP and block dataset. This is particularly relevant if there are substantial differences in the two genomic relationship matrices (e.g. use haplotype blocks for the epistatic interactions (Martini et al., 2017) and use SNPs for additive effects). The results in this section indicate some potential of the haplotype block framework to improve prediction, but more work is required to investigate appropriate parameter settings and utilize more information contained in the blocks besides simple presence and absence.

5.3 Design of breeding programs

In this section, potential breeding schemes that can be simulated via the R-package MoBPS (R Core Team, 2017; Pook et al., 2018) are showcased. All examples listed here correspond to those in the user manual of MoBPS (Supplementary B). In addition, potential analyses that can be performed based on these simulations are briefly discussed.

Although given simulations are based on cohorts, internally all simulations are performed on the base of individuals, allowing for a high flexibility in design and opportunities in a subsequent analysis. Note that all results shown in this section are based on a single simulation (using *set.seed(1)*). For extended evaluation and comparison between breeding schemes and methods, more replicates are highly recommended and, in contrast to real-world applications, can be provided in the setting of a simulation study (Chapter 1.5).

5.3.1 Introgression of a single QTL in chicken

Introducing genetic variation that is not present in an elite line is common practice in a breeding program. Among others, other elite lines, landraces or material from gene banks (La Mara et al., 2013; McCouch et al., 2012) can be used to increase the genetic variation. Variation can be increased in terms of overall genetic variance but also in regard to a local variant with positive features like resistance to specific diseases. In the following, programming code is given to simulate a simple introgression scheme with a single beneficial QTL, as used in similar form in Ha et al. (2017) for the introgression of a blue eggshell QTL in chicken. Note that in this example no traditional SNPs are used but instead it is just distinguished between the genetic origin from the elite line (0) and the wild population (1):

```
50 # Generate an input SNP-dataset
51 # 10 White-Layer (0) (20 haplotypes, 5'000 SNPs)
52 # 10 Wild population (1) (20 haplotypes, 5'000 SNPs)
53 dataset1 <- matrix(0, nrow = 5000, ncol = 20)
54 dataset2 <- matrix(1, nrow = 5000, ncol = 20)
55
```

```
# Generation of a trait
# Colums code: SNP, chromosome, effect 00, effect 01, effect 11
# Blue Eggshell QTL is positioned on SNP 2000, chromosome 1
major_qtl <- c(2000, 1, 0, 10000, 20000)
# In all other positions the white layer genome is assumed to be favorable
# All marker effects combiened are smaller than the blue eggshell QTL
rest <- cbind(1:5000, 1, 1, 0.5, 0)
trait <- rbind(major_qtl, rest)

# Generation of the base-population
# First 10 individuals are female (sex=2)
# Next 10 individuals are male (sex=1)
population <- creating.diploid(dataset = cbind(dataset1, dataset2),
                               real.bv.add = trait, name.cohort = "Founders",
                               sex.s = c(rep(2,10), rep(1,10)))

# Simulate random mating:
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(10,10),
                               best1.from.cohort = "Founders_M",
                               best2.from.cohort = "Founders_F",
                               name.cohort = "F1")

# Simuation of matings with selection:
# Top 50 cocks are mated to the 10 founder hens
# Selection of the cocks based on their genomic value ("bv")
# Target: Increase share of white layer while preserving blue egg shell QTL

population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(50,10),
                               best1.from.cohort = "F1_M",
                               best2.from.cohort = "Founders_F",
                               name.cohort = "BC1", selection.m = "function",
                               selection.criteria.type = "bv")
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(50,10),
                               best1.from.cohort = "BC1_M",
                               best2.from.cohort = "Founders_F",
                               name.cohort = "BC2", selection.m = "function",
                               selection.criteria.type = "bv")
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(50,10),
                               best1.from.cohort = "BC2_M",
                               best2.from.cohort = "Founders_F",
                               name.cohort = "BC3", selection.m = "function",
                               selection.criteria.type = "bv")

# Mating of cocks and hens that are heterozygous in blue egg shell QTL
# 25% of resulting offspring should be homozygous in blue egg shell QTL

population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(50,50),
                               best1.from.cohort = "BC3_M",
                               best2.from.cohort = "BC3_F",
                               name.cohort = "IC", selection.m = "function",
                               selection.criteria.type = "bv")
```

As one would expect, the rate of genetic material originating from the wild population is much higher in the QTL region (Figure 5.3). Subsequently, different potential selection techniques could be compared to make the introgression scheme more efficient (Ha et al., 2017). Depending on the breeding objective efficiency stands for a variety of things, ranging from maintaining genetic diversity to increasing the share of material of the elite line to minimizing the number of individuals used to obtain a particular breeding objective.

5.3.2 Performing a cock rotation to avoid inbreeding in chicken

There are a variety of different techniques to preserve genetic variation, ranging from the storage of genetic material in gene banks to specific breeding schemes. One of these breeding methods is a cock rotation in chicken. Here, hens are kept in separated boxes and all hens of one box are mated to the same cock from a different

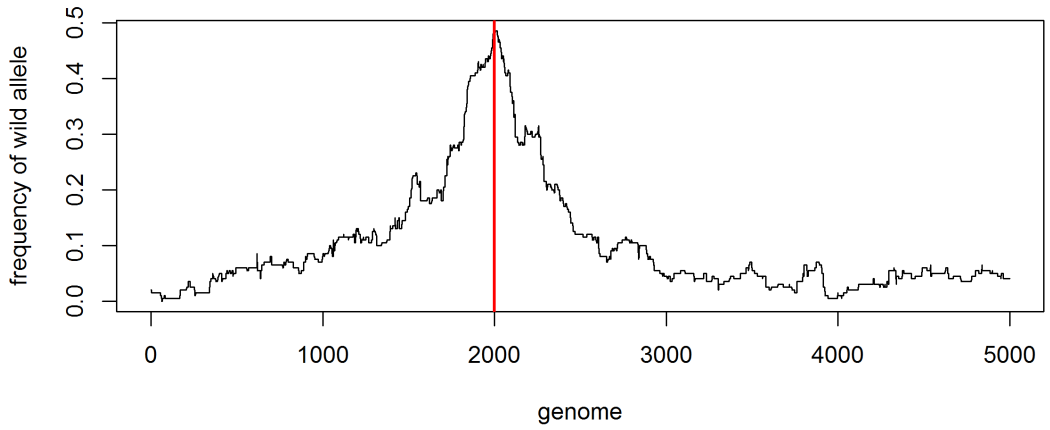


Figure 5.3: Frequency of genetic material stemming from the wild population after completion of the introgression scheme. The region of the QTL is indicated in red.

box. Exemplary, a program with seven boxes containing five hen and one cock each, can be simulated using MoBPS via the following programming code:

```
# Generate initial boxes with 5 hens (sex=2) and 1 cock (sex=1) each
population <- NULL
for(index in 1:7){
  population <- creating.diploid(population = population, nindi = 6,
                                 nsnp = 5000, sex.s = c(1, 2, 2, 2, 2, 2),
                                 name.cohort = paste0("Box_", index, "gen_0"))
}

# Simulate 25 generations of matings.
# Hens are rotated by one box per generation.
# best1.from.cohort is the cohort used as sires
# best2.from.cohort is the cohort used as dams
for(gen in 1:25){
    for(index in 1:7){
    population <- breeding.diploid(population, breeding.size = c(1,5),
                                   selection.size = c(1,5),
                best1.from.cohort = paste0("Box_",
                  if(index==1){7} else {index-1},"gen_", gen-1,"_M"),
                best2.from.cohort = paste0("Box_", index,"gen_", gen-1,"_F"),
                name.cohort = paste0("Box_", index, "gen_", gen),
                add.gen=gen+1
    )
  }
}

# Generate a population of same size without cock rotation
pop1 <- creating.diploid(nindi = 42, nsnp = 5000,
                         sex.s = c(rep(1,7), rep(2,35)))

# Simulate 25 generations of random mating
for(gen in 1:25){
  pop1 <- breeding.diploid(pop1, breeding.size = c(7,35),
                           selection.size = c(7,35))
}
```

The inbreeding levels in terms of IBD (Donnelly, 1983) when using a cock rotation are lower than in a random mating setup with the same number of animals (Figure 5.4). This can also be confirmed by deterministic formulas (Pook et al., 2017a). For further comparisons of more sophisticated mating schemes and deterministic formulas to derive expected inbreeding levels the interested reader is referred to Pook et al. (2017a,b).

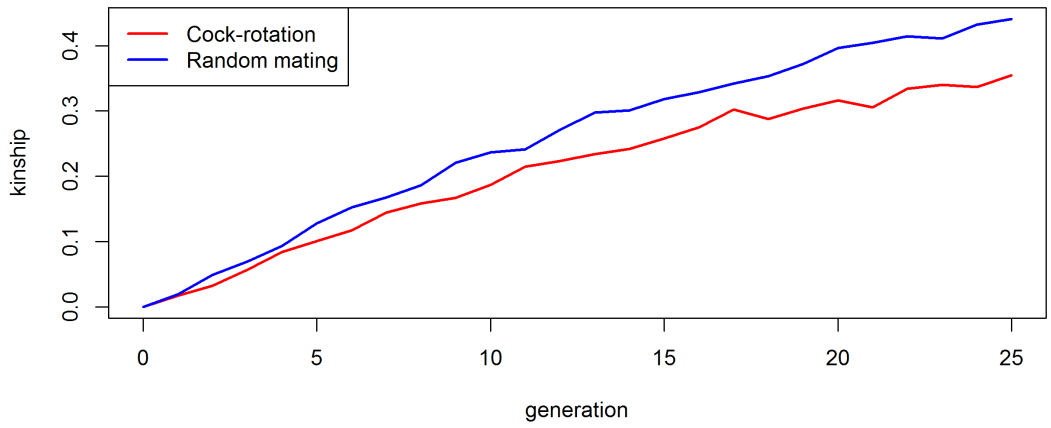


Figure 5.4: Realized kinship when using a cock rotation and a random mating breeding scheme with the same number of chicken.

5.3.3 Gene editing in a cow breeding program

With the development of biotechnologies like zinc-finger nucleases (ZFNs, (Carroll, 2011)), transcription activator-like effector nucleases (TALEN, (Bogdanove and Voytas, 2011)) and clustered regularly interspaced short palindromic repeats systems (CRISPR, (Jinek et al., 2012)) to modify specific loci of the DNA, potential use for breeding in both plants (Shan et al., 2014) and animals (Jenko et al., 2015) is of great interest. For the simulation of an exemplary breeding program that is utilizing such biotechnologies the following script can be used:

```
146 # Generation of a base population:
147 # 1'000 Founder individuals
148 # 5'000 SNPs
149 # 100 additive single marker QTL
150 population <- creating.diploid(nindi = 1000, nsnp = 5000,
151                                 n.additive = 100, name.cohort = "Founders")
152
153 # Simulation of a random mating generation
154 # 100 bulls (sex=1), 1'000 cows (sex=2) are generated
155 population <- breeding.diploid(population, breeding.size = c(100,1000),
156                                 selection.size = c(500,500),
157                                 best1.from.cohort = "Founders_M",
158                                 best2.from.cohort = "Founders_F",
159                                 name.cohort = "Random")
160
161 # Generate 200 offspring of both from the top 5 bulls / 200 cows
162 # Heritability of the trait is set to 0.5
163 # only phenotypes previously unobserved cows are generated
164 population <- breeding.diploid(population, breeding.size = 200,
165                                 selection.size = c(5,200), bve = TRUE,
166                                 heritability = 0.5, new.bv.observation = "non_obs_f",
167                                 selection.m = "function", name.cohort = "Top",
168                                 best1.from.cohort = "Random_M",
169                                 best2.from.cohort = "Random_F")
170
171 # Generate additional cows using all cows of the previous generation
172 # Cows are added to the same generation as the previous simulation
173 population <- breeding.diploid(population, breeding.size = c(0,900),
174                                 selection.size = c(5,1000),
175                                 selection.m = "function", name.cohort = "Sec_F",
176                                 best1.from.cohort = "Random_M",
177                                 best2.from.cohort = "Random_F",
178                                 use.last.sigma.e = TRUE,
179                                 add.gen = 3)
180
```

```
# Same cycle as before with additional genome editing
# Edits are chosen based on highest effects in rrBLUP
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(5,200), bve = TRUE,
                               new.bv.observation = "non_obs_f",
                               selection.m = "function",
                               name.cohort = "Top_Edit",
                               best1.from.cohort = "Top_M",
                               best2.from.cohort = c("Top_F","Sec_F"),
                               nr.edits = 20, estimate.u = TRUE,
                               use.last.sigma.e = TRUE)

population <- breeding.diploid(population, breeding.size = c(0,900),
                               selection.size = c(5,1000),
                               selection.m = "function", name.cohort = "Sec_Edit",
                               best1.from.cohort = "Top_M",
                               best2.from.cohort = c("Top_F","Sec_F"),
                               use.last.sigma.e = TRUE, add.gen = 4)
```

The potential of genetic improvement via genome editing is closely linked to the ability to identify the edits that need to be performed to increase the genomic value of the offspring. In case of additive effects of single markers this implies the identification of causal variants. For a detailed discussion on the potential use of genome editing techniques in practice, the interested reader is referred to Chapter 5.4.

5.3.4 Generation of MAGIC population in maize

Especially for crops it is common practice to generate multi-parent advanced generation inter-cross (MAGIC) populations to improve the power of QTL mapping and variety development (Bandillo et al., 2013). The following script can be used to simulate the generation of a MAGIC population in MoBPS (R Core Team, 2017; Pook et al., 2018). A similar design was also used in Chapter 2 to derive the ability of HaploBlocker to recover founder haplotypes.

```
# Generation of 20 fully-homozygous founders lines
# All plants are stored as male individuals (sex=0)
population <- creating.diploid(nindi = 20, sex.quota = 0 , template.chip = "maize" ,
                               dataset = "homorandom", name.cohort = "F0")

# Simulate matings between all founders.
# Each plan is involved in exactly 19 matings.
population <- breeding.diploid(population, breeding.size = c(190,0),
                               breeding.all.combination = TRUE,
                               selection.size = c(20,0),
                               best1.from.cohort = "F0", name.cohort = "F1")

# Simulate matings between plants of the last generation.
# Each plant is involved in exactly 2 matings.

population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = c(2,0),
                               best1.from.cohort = "F1", name.cohort = "F2")
population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = c(2,0),
                               best1.from.cohort = "F2", name.cohort = "F3")
population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = c(2,0),
                               best1.from.cohort = "F3", name.cohort = "F4")
```

5.3.5 Generation of a base population with LD and a hard sweep

Unless a real dataset is imported in MoBPS, markers in the first generation are not linked. In order to perform analysis on a dataset with association between physically linked markers one can consider generating such data in tools like MaCS (Chen et al., 2009) and QMSim (Sargolzaei and Schenkel, 2009) or first simulating a couple of generations of matings to generate these associations. A simulation to build up LD in MoBPS (R Core Team, 2017; Pook et al., 2018) can be performed as follows:

```
# Generate a starting population with 5000 SNPs and 200 individuals
# and a single chromosome of length 2 Morgan.
population <- creating.diploid(nsnp = 5000, nindi = 200, chromosome.length = 2)

# LD build up via 100 generations of random mating
# Each generation contains 200 individuals
for(index in 1:100){
  population <- breeding.diploid(population, breeding.size = 200,
                                 selection.size = c(100,100))
}

# Derive allele frequency and check LD for the last generation:
genotype.check <- get.geno(population, gen = length(population$breeding))
p_i <- rowMeans(genotype.check)/2
ld.decay(population, genotype.dataset = genotype.check, step = 10, max = 500)
```

As shown by the LD-decay (Figure 5.5), significant associations between physically linked markers arise after 100 generations of random mating. The simulation of a base population can be made arbitrarily more complicated to account for the specific population history one wants to work with. In the following, exemplary programming code to generate a single hard sweep (Walsh and Lynch, 2018) is given:

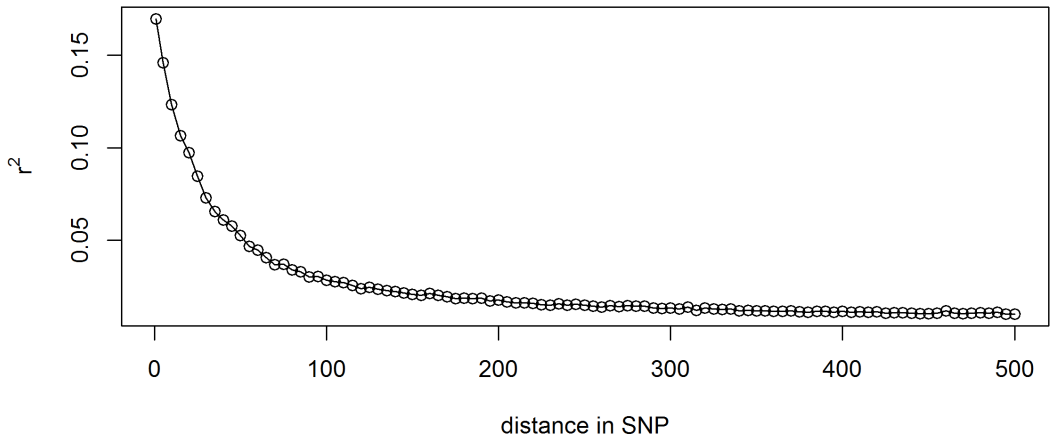


Figure 5.5: LD-decay of the base population after 100 generations of random mating via the MoBPS utility function *ld.decay()*.

```
# Simulate a favorable mutation in a previously fixed marker
fixated_markers <- which(p_i==0) # Which markers are fixated
qtl_posi <- sample(fixated_markers, 1) # Selected a fixated marker in A
trait <- cbind(qtl_posi, 1, 0, 1, 2) # SNP, Chromosome, Effect AA, Effect AB, Effect BB
population <- creating.trait(population, real.bv.add = trait)

# Generate a mutation in the first male individual
```

```
population <- mutation.intro(population, 101, 1, 1, qtl_posi)

# Simulate generations with selection pressure
# Individuals with the favorable SNP are picked 5 times as often
for(index in 1:25){
  population <- breeding.diploid(population, breeding.size = 200,
                                 selection.size = c(100,100),
                                 best.selection.ratio.m = 5,
                                 best.selection.ratio.f = 5)
}
```

The allele frequency at the location of the hard sweep over time can be analyzed via the MoBPS utility function *analyze.population()* (Figure 5.6, Supplementary B). The analysis of the dataset of the last generation of this simulation reveals structural differences in the sweep area. IHH scores based on EHH (Sabeti et al., 2002; Voight et al., 2006) and bEHH (Chapter 2) show clear signs of selection (Figure 5.7). Both IHH curves are very similar, showing that the block based approach is a good approximation of the traditional approach. In case the simulated mutation rate is drastically increased (e.g. $1 \cdot 10^{-3}$), IHH scores are reduced in both cases (Figure 5.8).

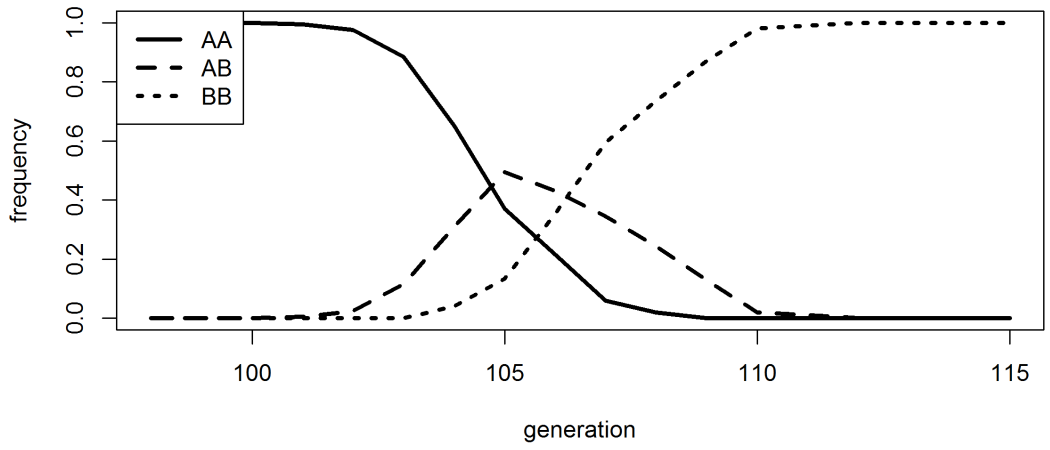


Figure 5.6: Plot of the allele frequency in the QTL undergoing a hard sweep via the MoBPS utility function *analyze.population()*. Allele B is favorable and the first mutation is simulated to occur in generation 101.

In studies like Islam et al. (2016) it is commonly observed that the use of haplotype blocks can increase the mapping power for QTLs. The simulation performed there shows slightly more pronounced peaks in the region of the hard sweep but more research and repetitions are needed to draw statistically sound conclusions for this.

5.4 Potential of gene editing in breeding

In this section, the potential of gene editing for quantitative traits in breeding is discussed. In this context, Jenko et al. (2015) proposed an approach for the promotion

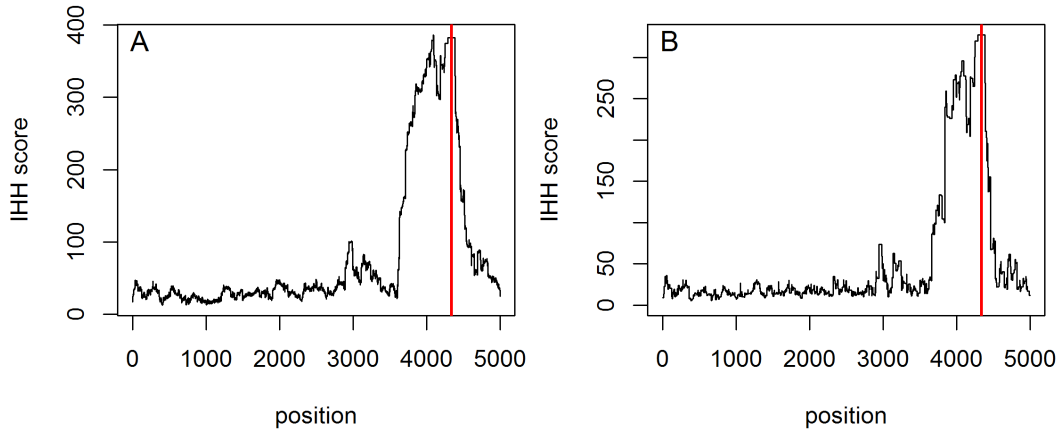


Figure 5.7: IHH scores of selection based on EHH (A) and bEHH (B). The location of the sweep is indicated in red.

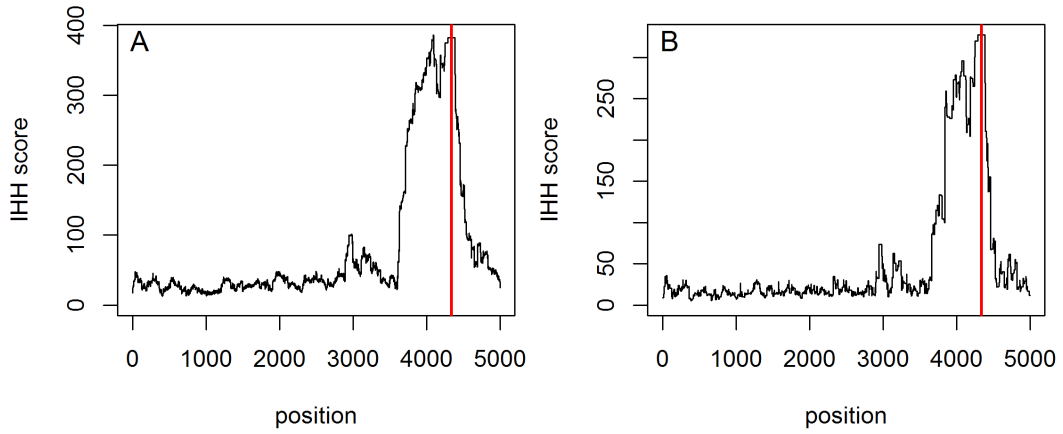


Figure 5.8: IHH scores of selection based on EHH (A) and bEHH (B) for an increased mutation rate. The location of the sweep is indicated in red.

of alleles by genome editing (PAGE) to accelerate the genetic progress in a cow breeding program and results were critically analyzed by Simianer et al. (2018). In this section, the manuscript by Simianer et al. (2018) is given first. Subsequently, the general usefulness and potential of genome editing in breeding for both livestock and crops is discussed.

5.4.1 Turning the PAGE - the potential of genome editing in breeding for complex traits revisited

This manuscript is a joint work of Henner Simianer[1], Torsten Pook[1] and Martin Schlather[2] and was presented at the *World Congress on Genetics Applied to Livestock Production 2018*. For reasons of uniformity in this thesis, the conference/journal style is not used in this section.

1: University of Goettingen, Animal Breeding and Genetics Group, Albrecht-Thaer-Weg 3, 37075 Goettingen, Germany
2: School of Business Informatics and Mathematics, University of Mannheim, A5, 68131 Mannheim, Germany

Author contributions by TP

TP performed the simulations for the study, contributed in the analysis and participated in revision of the manuscript.

Summary

In a recent study, Jenko et al. (2015) proposed to accelerate genetic progress by integrating a genome editing (GE) step in genomic breeding programs. This concept, called "promotion of alleles by genome editing" (PAGE) was implemented in a simulation study suggesting a substantial extra increase of genetic gain. As an example, editing in each generation the top 25 sires at the 20 quantitative trait nucleotides (QTN) with the highest effect was found to increase the genetic progress by 100% compared to genomic selection alone. We conducted a complex simulation study in which selection was on estimated GBLUP breeding values, the causal QTN were assumed unknown, and SNPs to be edited were identified by statistical means from the simulated data. We found the extra genetic progress due to PAGE to be be-tween 2 and 20 per cent, thus only a fraction of what was reported by Jenko et al. (2015). The observed difference is mainly attributed to the low power to detect true QTN. The best results were obtained with highly heritable traits, larger mapping populations and a limited number of true QTN, while performance was inferior both with low heritability traits and when QTN to be edited were identified by GWAS rather than by random regression BLUP. We argue that the true genetic architecture of complex traits will likely be much more complex than simulated here, which will further compromise the power of detecting true QTN and the predictability of the

effects of GE steps. These considerations together with the reported results indicate that overly optimistic expectations regarding the potential of PAGE should be taken with a pinch of salt.

Introduction

The concept of genome editing (GE) was first successfully demonstrated in mammals in the 1970s, but only had a breakthrough with the introduction of the CRISPR-Cas9 system (Jinek et al., 2012). This approach combines simplicity, high accuracy, high efficiency and limited off-target effects and has a considerable potential for multiplexing. Based on this technological perspective Jenko et al. (2015) proposed to accelerate genetic progress by integrating a genome editing step in genomic breeding programs. The basic idea was to augment a genomic dairy cattle breeding scheme as originally suggested by Schaeffer (2006) by changing a limited number of quantitative trait nucleotides (QTN) towards the most advantageous genotype in a given number of selected individuals via GE. This concept, called "promotion of alleles by genome editing" (PAGE), was demonstrated in a simulation study suggesting that, compared to a classical genomic selection (GS) scheme without a GE step, the expected response to selection with PAGE was between 1.08 and 4.12 times higher. As an example, editing in each generation the top 25 sires at the 20 QTN with the highest effect was found to double the genetic progress compared to GS alone.

While these results appear quite promising, a number of caveats must be made. Most importantly, it was assumed that the true effects of segregating QTN were known and could be used to identify those QTN with the largest effects to be edited in the top sires. Also, selection was assumed to be based on known true breeding values.

The aim of our study was to assess the expected benefit of PAGE with a somewhat more realistic scenario. For this, we set up a similar (but not completely identical) simulation scheme and used genomic best linear unbiased prediction (GBLUP, (VanRaden, 2008)) for the selection step and estimated SNP effects from random regression BLUP (RRBLUP, (Meuwissen et al., 2001)) or alternatively GWAS results to identify the top QTNs to be edited.

Simulation scheme

We started with a high density SNP data set from a real dairy cattle population which is assumed to reflect the major characteristics of a breeding population under selection w.r.t. linkage disequilibrium, allele frequency spectra etc.. In the reference scenario we generated a base population of 500 male and $N = 10'000$ female individuals. We selected 50'000 SNPs, of which 1000 SNPs (2%) were randomly assigned to be QTN with an additive effect drawn from a standard normal distribution. Phenotypes of females were calculated as the sum of the genotype values across all QTN plus a random number sampled from a normal distribution such that the heritability

had the desired value. We then used GBLUP to select the top 25 sires and 500 cows as potential bull sires (BS) and bull dams (BD), the remaining cows were used as cow dams (CD). In the first ten generations, the 25 BS were randomly mated to the 500 BD to produce 500 male selection candidates for the next generation. The 25 BS were also mated to the 10'000 females (BD and CD) to produce 10'000 female selection candidates for the next generation.

Starting from generation 11, the 25 selected BS underwent a genome editing step in the PAGE approach: first, effects were estimated for all SNPs using RRBLUP. Next, each of the 25 BS was genome edited in that he was made homozygous for the favourable allele at those loci which had the highest estimated absolute SNP effects and for which he wasn't already carrying the most beneficial genotype. These "edited bull sires" (eBS) were used in the same way as the BS in generation one to ten for ten more generations (11 – 20) (Figure 5.9)

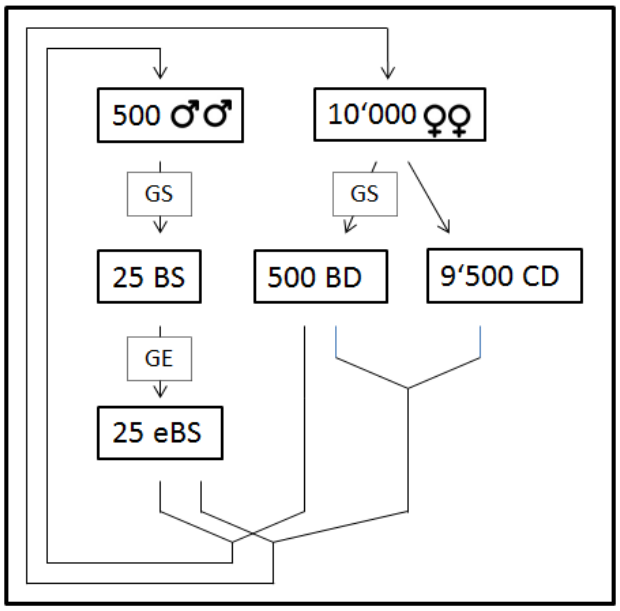


Figure 5.9: One generation of the PAGE selection scheme (reference scenario).

Starting from this reference scenario, a number of parameters were varied, one at a time (reference value underlined):

- the heritability: h^2= 0.05, 0.3, 0.6
- the number of true QTN: 1000, 500, 250
- the number of SNPs: 50'000, 100'000, 200'000
- the cow population size N: 10'000, 25'000
- selection of target SNPs for editing: RRBLUP or GWAS

For each scenario we generated and analysed 100 replicates. The main criterion of interest was the change of genetic progress from generation 11 to 20 when using the PAGE approach compared to using genomic selection without GE in the same generations. We also recorded the success rate of QTN identification as the proportion of edits that were made on true QTNs in each generation. The simulations were performed using the R-package RekomBre (Pook, unpublished) [1] on a server cluster with Intel E5-2670 (2X8 core 2.6 GHz) and AMD Opteron 6378 (4X16 2.4 GHz) processors.

Results and Discussion

Compared to selection based on GBLUP, we found that the PAGE approach led to an 11.6 per cent increase of the genetic progress in the reference scenario (Fig. 5.10 and 5.11), which is about one tenth of the improvement predicted by Jenko et al. (2015) for a similar scenario with 20 edits per top bull. Our implementation differs from the one described in Jenko et al. (2015) in many details, but we suspect two major causes for the observed discrepancy in the results: (i) while in Jenko et al. (2015) candidates are selected on their true breeding values, selection in our implementation is based on genomic breeding values, which are less accurate, but this affects selection based on GBLUP and PAGE both in a similar way; (ii) while loci to be edited are selected based on their true (but in reality unknown) effects in Jenko et al. (2015), selection is based on estimated allele effects from RRBLUP in our implementation. Especially the second discrepancy has a major effect, since in the first generation of PAGE only 10 per cent of the edits are done on real QTN (see. Fig. 5.10), i.e. on average just 2 of the 20 loci edited in a bull were on target, while 18 edits were made on non-causal SNPs and thus had no consequence for the true genetic values of the edited bulls. This success rate even drops to 6.2 per cent in generation 20, since the detectable large effect QTNs tend to become fixed in the first generations of PAGE and the remaining polymorphic QTNs in later generations have smaller effects and are thus even more difficult to detect.

Varying some of the parameters in the reference scenario yielded the following observations (Fig. 5.11): with a reduced number of QTN (500 or 250) the extra genetic progress due to PAGE increased to 14.5 and 15.5%, respectively, which can be explained by the relatively larger effects and thus better detectability of true QTN. However, with a smaller number of true QTN the success rate of QTN detection also tended to deteriorate faster. Increasing the number of SNPs had a slightly negative effect on the expected genetic progress. Increasing the population size to 25'000 cows led to a higher success rate in QTN detection and increased the extra genetic progress due to PAGE to 14.2%. Varying the heritability yielded the strongest effects: while doubling h^2 to 0.6 led to an extra genetic progress of 20.7%, the latter was only 4.9% with $h^2 = 0.05$. An even more dramatic effect was observed when SNPs to be edited were identified based on GWAS results rather than RRBLUP

[1] RekomBre was used as the internal development title of MoBPS (Chapter 4)

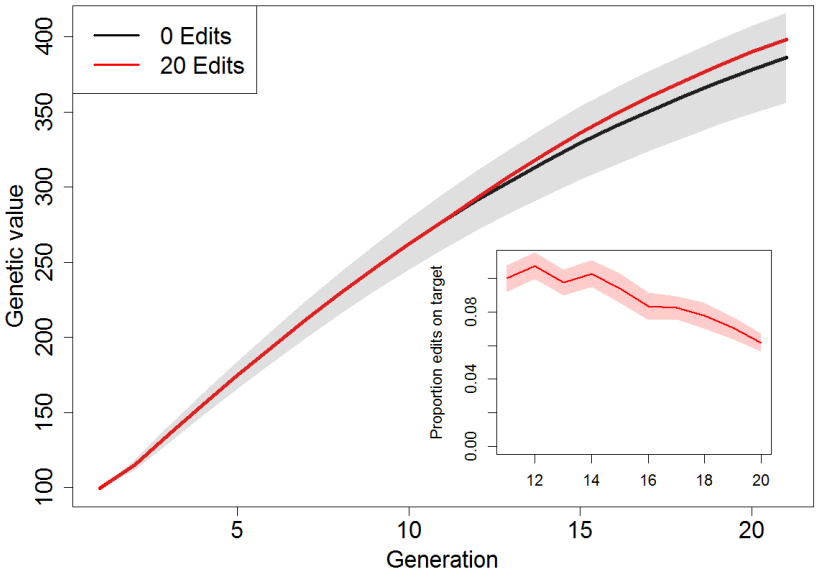


Figure 5.10: Genetic progress with PAGE (20 edits) vs. the pure GS variant (0 edits) and the proportion of edits on target in the GE generations (inner figure) in the reference scenario.

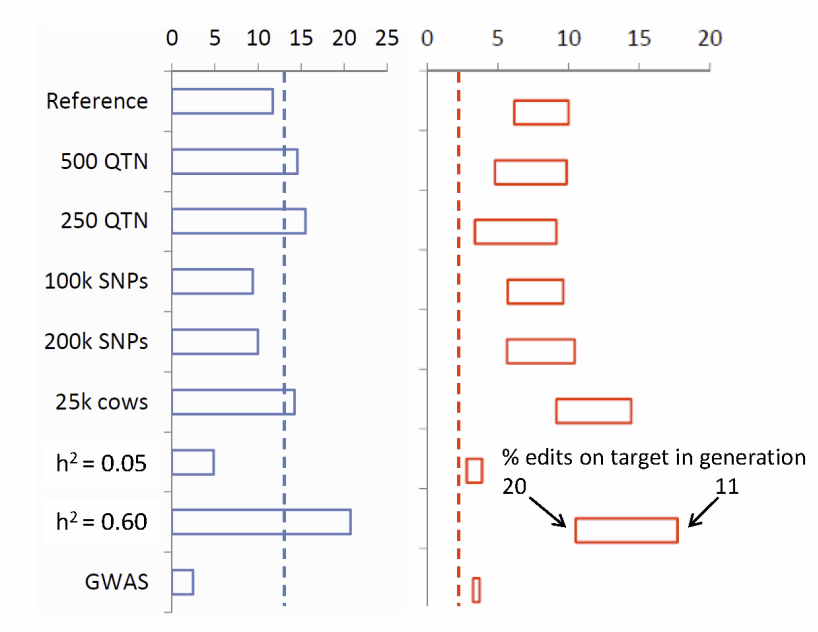


Figure 5.11: Extra genetic progress of PAGE vs. GS (left) and proportion of edits on target in generations 11 and 20 (right) for the different scenarios.

results. In this case, the proportion of edits of true QTN was between 3.2% and 3.7% and thus only marginally above the 2% expected when SNPs to be edited were selected completely at random, and consequently the extra genetic progress due to PAGE was only 2.5% in this scenario.

Conclusions

Across all scenarios studied, the increase of efficiency due to PAGE didn't even get close to the rates suggested by the simulation study of Jenko et al. (2015), which can be mainly attributed to the low success rate of identifying true QTN for the editing step. One might expect that with increasing numbers of genotyped individuals the power to detect causal QTN will increase. However, the "true" genetic model underlying our study as well as the one by Jenko et al. (2015) is heavily simplistic by assuming complete additivity. Genetic architecture of complex traits is expected to be far more complex in reality and will comprise epistatic interactions, complex nonlinear regulation processes and redundancies, genotype by environment and genotype by sex interactions etc. (Mackay, 2004). Causal variants will also be of more complex nature than being just single SNPs. All this will further reduce the power of finding true causal variants as targets for GE. Even if such a causal variant is detected and edited, the effect of this modification on the phenotype will be hardly predictable under such a complex genetic trait architecture. While GE presumably has some potential in breeding for monogenic traits, all these considerations together with the reported results indicate that overly optimistic expectations regarding the potential of PAGE in breeding for complex traits should be taken with a pinch of salt.

5.4.2 Further thoughts

In addition to the concerns already expressed in the previous subsection, there are further things to consider when it comes to the use of gene editing in breeding for quantitative traits. At the current state of research, any application of the CRISPR-Cas9 system (Jinek et al., 2012) will lead to off-target mutation. Especially for the use in animal and human genetics these off-target mutations can lead to an increased risk of introducing cancer (Zhang et al., 2015; Haapaniemi et al., 2018). Combined with the possibility of target regions not being successful edited (Hai et al., 2014), it has to be conclude that further improvements to editing technologies are necessary.

Furthermore, it must be taken into account that the functionality of a gene is usually far more complex than the assumed additive effects of single loci (Simianer, 2018). It should be noted that the computational techniques for QTL identification used here (rrBLUP, GWAS) do not utilize protein coding (Davydov et al., 2010; Siepel et al., 2005) or the like but achieving a significantly higher proportion of edits on target still seems unrealistic for quantitative traits today. It should be noted that in recent years, more and more work has been done to improve the functional understanding

of the genome (Washburn et al., 2019; Robinson, 2019) and fine-mapping of QTL (Kichaev et al., 2017). Genome editing itself can even be used as a tool to improve the identification process of functional variants (Knott and Doudna, 2018).

In addition to the scientific concerns, the application of biotechnologies in practice comes with problems of social acceptance and legal questions. In Europe the application of CRISPR-Cas9 leads to resulting plants being classified as genetically modified organisms (CJEU, 2018; Gelinsky and Hilbeck, 2018).

Nevertheless, there is considerable potential in the use of the CRISPR-Cas9 system in breeding. Instead of focusing on the biggest positive effects as in Jenko et al. (2015), one could instead specifically targeting deleterious variants as suggested in Johnsson et al. (2019) and thereby reduce the genetic load (Crnokrak and Barrett, 2002). These deleterious variants can be identified in a variety of methods (Daetwyler et al., 2014; Ramu et al., 2017; Mezmouk and Ross-Ibarra, 2014). Instead of focusing on quantitative traits one could instead consider using genome editing for simpler traits with limited number of loci like the polled locus in cattle (Mueller et al., 2019). In plant breeding there are even examples of genome editing procedures that are already implemented in breeding (Waltz, 2018; Zsögön et al., 2018).

5.5 Outlook & Conclusions

5.5.1 Outlook for HaploBlocker

The main focus of the development of HaploBlocker in this thesis was the establishment of the methodology itself. Even though some first applications were shown, more research is needed to fully exploit the potential of the haplotype block derived in HaploBlocker.

Potential fields of applications for HaploBlocker in quantitative genetics are manifold. Most existing methods that use SNPs need to be slightly modified to use haplotype blocks instead of SNPs. It should be noted that the block structure in HaploBlocker differs substantially from traditional haplotype block approaches. Among others, more sophisticated approaches for the detection of selection signatures in tests as suggested by Beissinger et al. (2018) and genomic prediction (Chapter 5.2, (Meuwissen et al., 2014; Jiang et al., 2018)) can be considered here. In addition, haplotype blocks can potentially be used for the imputation of sequence data by merging reads of haplotypes with the same haplotype block. In contrast to commonly used methods like Li and Stephens (2003); Howie et al. (2009); Das et al. (2016); Browning et al. (2018) the read depth in sequence data could be utilized by this and potentially even be increased.

5.5.2 Outlook for MoBPS

Both the data storage and the modular design to carry out simulations in MoBPS provide a lot of flexibility and thus enables a multitude of possible extensions to the simulation framework. Even though the user can, in principle, already use his own estimation techniques, commonly applied methods like single-step GBLUP (Aguilar et al., 2010) have to be implemented in a computationally efficient way. In particular for the use in plant genetics the inclusion of environmental effects and all available and upcoming breeding technologies need proper implementation.

The data storage of haplotypes is highly efficient, thus enabling the simulation of full sequence data to potentially account for more complex effect structures and simulate genetic effects that reflect actual gene/genome functionality better.

In addition to the technical side of the implementation, the use of MoBPS for scientists and breeders with limited programming background is a key concern. A web-based application for MoBPS is currently under development to achieve the required user-friendliness. This is a joint project of Torsten Pook, Amudha Ganesan, Ngoc-Thuy Ha, Lisa Büttgen and Henner Simianer (all: Department of Animal Sciences, Center for Integrated Breeding Research, University of Goettingen, Goettingen, 37075, Germany). The interested reader is referred the user manual of MoBPS for details on this (Supplementary B and `https://github.com/tpook92/MoBPS`).

5.5.3 Concluding remarks

HaploBlocker and MoBPS provide powerful tools for breeding and quantitative genetics. Both tools are designed to process datasets with a high number of individuals and markers, and computation times for most potential bottlenecks scale linearly in both dimensions. HaploBlocker and MoBPS therefore provide valuable tools to assist in solving large scale problems of high complexity.

In addition, both tools provide the flexibility to customize the algorithm and design space. For HaploBlocker, this implies tuning of the input parameter to adjust the structure of obtained haplotype library according to the individual needs for subsequent analyses. For MoBPS, this flexibility not only allows for a flexible design of the breeding program itself but also the modification of certain steps of the simulation procedure to test own methodology such as a new estimator for breeding values. A potential method that could be considered for this purpose are artificial neural networks (ANN, (Bellot et al., 2018; Pook, 2019a)). ANN not only have the potential to improve the breeding value estimation in terms of accuracy but also have the decisive advantage of linearly scaling in the number of individuals and thus offer further opportunities for the field of big data analysis in general.

6 References

1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015. ISSN 0028-0836.

Aguilar, I , Misztal, I , Johnson, D. L , Legarra, A , Tsuruta, S , and Lawlor, T. J . Hot topic: A unified approach to utilize phenotypic, full pedigree, and genomic information for genetic evaluation of holstein final score. *Journal of Dairy Science*, 93(2):743–752, 2010. ISSN 0022-0302.

Akdemir, D , Jannink, J.-L , and Isidro-Sánchez, J . Locally epistatic models for genome-wide prediction and association by importance sampling. *Genetics Selection Evolution*, 49(1):74, 2017. ISSN 1297-9686.

Albrecht, T , Wimmer, V , Auinger, H.-J , Erbe, M , Knaak, C , Ouzunova, M , Simianer, H , and Schön, C.-C . Genome-based prediction of testcross values in maize. *Theoretical and Applied Genetics*, 123(2):339, 2011. ISSN 0040-5752.

Albrechtsen, A , Nielsen, F. C , and Nielsen, R . Ascertainment biases in snp chips affect measures of population divergence. *Molecular biology and evolution*, 27(11): 2534–2547, 2010. ISSN 1537-1719.

Bandillo, N , Raghavan, C , Muyco, P. A , Sevilla, M. A. L , Lobina, I. T , Dilla-Ermita, C. J , Tung, C.-W , McCouch, S , Thomson, M , and Mauleon, R . Multi-parent advanced generation inter-cross (magic) populations in rice: Progress and potential for genetics research and breeding. *Rice*, 6(1):11, 2013. ISSN 1939-8433.

Barrett, J. C , Fry, B , Maller, J , and Daly, M. J . Haploview: analysis and visualization of ld and haplotype maps. *Bioinformatics*, 21(2):263–265, 2005. ISSN 1367-4803.

Baum, L. E and Petrie, T . Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966. ISSN 0003-4851.

Beissinger, T , Kruppa, J , Cavero, D , Ha, N.-T , Erbe, M , and Simianer, H . A simple test identifies selection on complex traits. *Genetics*, 209(1):321–333, 2018. ISSN 0016-6731.

Bellot, P , de Los Campos, G , and Pérez-Enciso, M . Can deep learning improve genomic prediction of complex human traits? *Genetics*, 210(3):809–819, 2018. ISSN 0016-6731.

Bellott, D. W , Skaletsky, H , Pyntikova, T , Mardis, E. R , Graves, T , Kremitzki, C , Brown, L. G , Rozen, S , Warren, W. C , and Wilson, R. K . Convergent evolution of chicken z and human x chromosomes by expansion and gene acquisition. *Nature*, 466(7306):612, 2010. ISSN 0028-0836.

Bogdanove, A. J and Voytas, D. F . Tal effectors: Customizable proteins for dna targeting. *Science*, 333(6051):1843–1846, 2011. ISSN 0036-8075.

Boichard, D , Chung, H , Dassonneville, R , David, X , Eggen, A , Fritz, S , Gietzen, K. J , Hayes, B. J , Lawley, C. T , and Sonstegard, T. S . Design of a bovine low-density snp array optimized for imputation. *PLOS ONE*, 7(3):e34130, 2012. ISSN 1932-6203.

Bradbury, P. J , Zhang, Z , Kroon, D. E , Casstevens, T. M , Ramdoss, Y , and Buckler, E. S . Tassel: Software for association mapping of complex traits in diverse samples. *Bioinformatics*, 23(19):2633–2635, 2007. ISSN 1367-4803.

Branton, D , Deamer, D. W , Marziali, A , Bayley, H , Benner, S. A , Butler, T , Di Ventra, M , Garaj, S , Hibbs, A , and Huang, X . The potential and challenges of nanopore sequencing. *Nanoscience and technology: A collection of reviews from Nature Journals*, pages 261–268, 2010.

Browning, B. L and Browning, S. R . Efficient multilocus association testing for whole genome association studies using localized haplotype clustering. *Genetic Epidemiology*, 31(5):365–375, 2007. ISSN 1098-2272.

Browning, B. L and Browning, S. R . A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. *The American Journal of Human Genetics*, 84(2):210–223, 2009. ISSN 0002-9297.

Browning, B. L and Browning, S. R . A fast, powerful method for detecting identity by descent. *The American Journal of Human Genetics*, 88(2):173–182, 2011. ISSN 0002-9297.

Browning, B. L and Browning, S. R . Improving the accuracy and efficiency of identity-by-descent detection in population data. *Genetics*, 194(2):459–471, 2013. ISSN 0016-6731.

Browning, B. L and Browning, S. R . Genotype imputation with millions of reference samples. *The American Journal of Human Genetics*, 98(1):116–126, 2016. ISSN 0002-9297.

Browning, B. L , Zhou, Y , and Browning, S. R . A one-penny imputed genome from next-generation reference panels. *The American Journal of Human Genetics*, 103 (3):338–348, 2018. ISSN 0002-9297.

Browning, S. R . Multilocus association mapping using variable-length markov chains. *The American Journal of Human Genetics*, 78(6):903–913, 2006. ISSN 0002-9297.

Burrows, M and Wheeler, D. J . A block-sorting lossless data compression algorithm. *Technical report 124*, 1994, 1994.

Cabrera-Bosquet, L , Crossa, J , von Zitzewitz, J , Serret, M. D , and Luis Araus, J . High–throughput phenotyping and genomic selection: The frontiers of crop breeding converge f. *Journal of integrative plant biology*, 54(5):312–320, 2012. ISSN 1672-9072.

Carroll, D . Genome engineering with zinc-finger nucleases. *Genetics*, 188(4):773–782, 2011. ISSN 0016-6731.

Check Hayden, E . Technology: The $1,000 genome. *Nature News*, 507(7492):294, 2014.

Chen, G. K , Marjoram, P , and Wall, J. D . Fast and flexible simulation of dna sequence data. *Genome Research*, 19(1):136–142, 2009. ISSN 1088-9051.

Churchill, G. A , Airey, D. C , Allayee, H , Angel, J. M , Attie, A. D , Beatty, J , Beavis, W. D , Belknap, J. K , Bennett, B , and Berrettini, W . The collaborative cross, a community resource for the genetic analysis of complex traits. *Nature Genetics*, 36(11):1133, 2004.

CJEU. Judgement of the court (grand chamber): 25 july 2018 in case c-528/16, 2018. URL `http://curia.europa.eu/juris/document/document.jsf?text=&docid=207002&pageIndex=0&doclang=EN&mode=req&dir=&occ=first&part=1&cid=2838693`.

Covarrubias-Pazaran, G . Genome-assisted prediction of quantitative traits using the r package sommer. *PLOS ONE*, 11(6):e0156744, 2016. ISSN 1932-6203.

Crnokrak, P and Barrett, S. C. H . Perspective: Purging the genetic load: a review of the experimental evidence. *Evolution*, 56(12):2347–2358, 2002. ISSN 0014-3820.

Da, Y , Wang, C , Wang, S , and Hu, G . Mixed model methods for genomic prediction and variance component estimation of additive and dominance effects using snp markers. *PLOS ONE*, 9(1):e87666, 2014. ISSN 1932-6203.

Daetwyler, H. D , Capitan, A , Pausch, H , Stothard, P , van Binsbergen, R , Brøndum, R. F , Liao, X , Djari, A , Rodriguez, S. C , and Grohs, C . Whole-genome sequencing of 234 bulls facilitates mapping of monogenic and complex traits in cattle. *Nature Genetics*, 46(8):858, 2014.

Daly, M. J , Rioux, J. D , Schaffner, S. F , Hudson, T. J , and Lander, E. S . High-resolution haplotype structure in the human genome. *Nature Genetics*, 29 (2):229–232, 2001.

Danecek, P , Auton, A , Abecasis, G , Albers, C. A , Banks, E , DePristo, M. A , Handsaker, R. E , Lunter, G , Marth, G. T , and Sherry, S. T . The variant call format and vcftools. *Bioinformatics*, 27(15):2156–2158, 2011. ISSN 1367-4803.

Das, S , Forer, L , Schönherr, S , Sidore, C , Locke, A. E , Kwong, A , Vrieze, S. I , Chew, E. Y , Levy, S , and McGue, M . Next-generation genotype imputation service and methods. *Nature Genetics*, 48(10):1284, 2016.

Das, S , Abecasis, G. R , and Browning, B. L . Genotype imputation from large reference panels. *Annual review of genomics and human genetics*, 19:73–96, 2018. ISSN 1527-8204.

Davydov, E. V , Goode, D. L , Sirota, M , Cooper, G. M , Sidow, A , and Batzoglou, S . Identifying a high fraction of the human genome to be under selective constraint using gerp++. *PLoS computational biology*, 6(12):e1001025, 2010. ISSN 1553-7358.

de los Campos, G . What fraction of the information contained in an omic set can be explained by other omics? paper # 22987. *Plant and Animal Genome Conference, San Diego, California*, 2017.

Donnelly, K. P . The probability that related individuals share some section of genome identical by descent. *Theoretical population biology*, 23(1):34–63, 1983. ISSN 0040-5809.

Druet, T and Georges, M . A hidden markov model combining linkage and linkage disequilibrium information for haplotype reconstruction and quantitative trait locus fine mapping. *Genetics*, 184(3):789–798, 2010. ISSN 0016-6731.

Egger-Danner, C , Cole, J. B , Pryce, J. E , Gengler, N , Heringstad, B , Bradley, A , and Stock, K. F . Invited review: Overview of new traits and phenotyping strategies in dairy cattle with a focus on functional traits. *Animal*, 9(2):191–207, 2015. ISSN 1751-7311.

Eggertsson, H. P , Jonsson, H , Kristmundsdottir, S , Hjartarson, E , Kehr, B , Masson, G , Zink, F , Hjorleifsson, K. E , Jonasdottir, A , and Jonasdottir, A . Graphtyper enables population-scale genotyping using pangenome graphs. *Nature Genetics*, 49(11):1654, 2017.

Endelman, J. B . Ridge regression and other kernels for genomic selection with r package rrblup. *The Plant Genome*, 4(3):250–255, 2011. ISSN 1940-3372.

Falconer, D. S and Mackay, T. F . Introduction to quantitative genetics. longman. *Essex, England*, 1996.

Fan, J , Han, F , and Liu, H . Challenges of big data analysis. *National Science Review*, 1(2):293–314, 2014. ISSN 2095-5138.

Faux, A.-M , Gorjanc, G , Gaynor, R. C , Battagin, M , Edwards, S. M , Wilson, D. L , Hearne, S. J , Gonen, S , and Hickey, J. M . Alphasim: Software for breeding program simulation. *The Plant Genome*, 9(3):doi:10.3835/plantgenome2016.02.0013, 2016. ISSN 1940-3372.

Fisher, R. A . Xv.—the correlation between relatives on the supposition of mendelian inheritance. *Earth and Environmental Science Transactions of the Royal Society of Edinburgh*, 52(2):399–433, 1918. ISSN 2053-5945.

Forsberg, S. K. G , Bloom, J. S , Sadhu, M. J , Kruglyak, L , and Carlborg, Ö . Accounting for genetic interactions improves modeling of individual quantitative trait phenotypes in yeast. *Nature Genetics*, 49(4):497, 2017.

Gabriel, S. B , Schaffner, S. F , Nguyen, H , Moore, J. M , Roy, J , Blumenstiel, B , Higgins, J , DeFelice, M , Lochner, A , and Faggart, M . The structure of haplotype blocks in the human genome. *Science*, 296(5576):2225–2229, 2002. ISSN 0036-8075.

Ganal, M. W , Durstewitz, G , Polley, A , Bérard, A , Buckler, E. S , Charcosset, A , Clarke, J. D , Graner, E.-M , Hansen, M , and Joets, J . A large maize (zea mays l.) snp genotyping array: development and germplasm genotyping, and genetic mapping to compare with the b73 reference genome. *PLOS ONE*, 6(12):e28334, 2011. ISSN 1932-6203.

Gautier, M and Vitalis, R . rehh: An r package to detect footprints of selection in genome-wide snp data from haplotype structure. *Bioinformatics*, 28(8):1176–1177, 2012. ISSN 1367-4803.

Geibel, J , Weigend, S , Weigend, A , Reimer, C , Pook, T , and Simianer, H . Array design and snp ascertainment bias. *World Congress on Genetics Applied to Livestock*, page 650, 2018.

Gelinsky, E and Hilbeck, A . European court of justice ruling regarding new genetic engineering methods scientifically justified: A commentary on the biased reporting about the recent ruling. *Environmental Sciences Europe*, 30(1):52, 2018. ISSN 2190-4715.

Gianola, D and Rosa, G. J. M . One hundred years of statistical developments in animal breeding. *Annu. Rev. Anim. Biosci.*, 3(1):19–56, 2015. ISSN 2165-8102.

Groenen, M. A , Wahlberg, P , Foglio, M , Cheng, H. H , Megens, H.-J , Crooijmans, R. P , Besnier, F , Lathrop, M , Muir, W. M , and Wong, G. K.-S . A high-density snp-based linkage map of the chicken genome reveals sequence features correlated with recombination rate. *Genome Research*, 19(3):510–519, 2009. ISSN 1088-9051.

Ha, N.-T , Pook, T , Dierks, C , Weigend, S , Preisinger, R , and Simianer, H . A simulation approach to optimize breeding programs with application to the introgression of the blue egg color into a high performing layer line. *10th European Symposium on Poultry Genetics*, 2017:107, 2017.

Haapaniemi, E , Botla, S , Persson, J , Schmierer, B , and Taipale, J . Crispr–cas9 genome editing induces a p53-mediated dna damage response. *Nature medicine*, 24(7):927, 2018.

Habier, D , Fernando, R. L , and Dekkers, J. C. M . The impact of genetic relationship information on genome-assisted breeding values. *Genetics*, 177(4):2389–2397, 2007. ISSN 0016-6731.

Hai, T , Teng, F , Guo, R , Li, W , and Zhou, Q . One-step generation of knockout pigs by zygote injection of crispr/cas system. *Cell research*, 24(3):372, 2014. ISSN 1748-7838.

Hayes, B and Goddard, M . Genome-wide association and genomic selection in animal breeding. *Genome*, 53(11):876–883, 2010. ISSN 0831-2796.

Hayes, B. J , Bowman, P. J , Chamberlain, A. J , and Goddard, M. E . Invited review: Genomic selection in dairy cattle: Progress and challenges. *Journal of Dairy Science*, 92(2):433–443, 2009. ISSN 0022-0302.

He, D . Ibd-groupon: an efficient method for detecting group-wise identity-by-descent regions simultaneously in multiple individuals based on pairwise ibd relationships. *Bioinformatics*, 29(13):i162–i170, 2013. ISSN 1367-4803.

He, S , Reif, J. C , Korzun, V , Bothe, R , Ebmeyer, E , and Jiang, Y . Genome-wide mapping and prediction suggests presence of local epistasis in a vast elite winter wheat populations adapted to central europe. *Theoretical and Applied Genetics*, 130(4):635–647, 2017. ISSN 0040-5752.

Henderson, C. R . Best linear unbiased estimation and prediction under a selection model. *Biometrics*, pages 423–447, 1975.

Henryon, M , Berg, P , and Sørensen, A. C . Animal-breeding schemes using genomic information need breeding plans designed to maximise long-term genetic gains. *Livestock Science*, 166:38–47, 2014. ISSN 1871-1413.

Herrera-Marcos, L , Lou-Bonafonte, J , Arnal, C , Navarro, M , and Osada, J . Transcriptomics and the mediterranean diet: A systematic review. *Nutrients*, 9 (5):472, 2017.

Heslot, N , Jannink, J.-L , and Sorrells, M. E . Perspectives for genomic selection applications and research in plants. *Crop Science*, 55(1):1–12, 2015.

Hickey, J. M , Kinghorn, B. P , Tier, B , Wilson, J. F , Dunstan, N , and van der Werf, J. H. J . A combined long-range phasing and long haplotype imputation method to impute phase for snp genotypes. *Genetics Selection Evolution*, 43(1): 12, 2011. ISSN 1297-9686.

Hickey, J. M , Crossa, J , Babu, R , and de los Campos, G . Factors affecting the accuracy of genotype imputation in populations from several maize breeding programs. *Crop Science*, 52(2):654–663, 2012.

Hill, W. G . Prediction and evaluation of response to selection with overlapping generations. *Animal Science*, 18(2):117–139, 1974. ISSN 1748-748X.

Hölker, A. C , Mayer, M , Presterl, T , Bolduan, T , Bauer, E , Ordas, B , Brauner, P. C , Ouzunova, M , Melchinger, A. E , and Schön, C.-C . European maize landraces made accessible for plant breeding and genome-based studies. *Theoretical and Applied Genetics*, pages 1–13, 2019. ISSN 0040-5752.

Holsinger, K. E and Weir, B. S . Genetics in geographically structured populations: Defining, estimating and interpreting f(st). *Nature Reviews Genetics*, 10(9):639, 2009. ISSN 1471-0056.

Horgan, R. P and Kenny, L. C . 'omic'technologies: Genomics, transcriptomics, proteomics and metabolomics. *The Obstetrician & Gynaecologist*, 13(3):189–195, 2011. ISSN 1744-4667.

Howie, B , Fuchsberger, C , Stephens, M , Marchini, J , and Abecasis, G. R . Fast and accurate genotype imputation in genome-wide association studies through pre-phasing. *Nature Genetics*, 44(8):955, 2012.

Howie, B. N , Donnelly, P , and Marchini, J . A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. *PLOS Genetics*, 5(6):e1000529, 2009. ISSN 1553-7404.

International Chicken Genome Sequencing Consortium. Sequence and comparative analysis of the chicken genome provide unique perspectives on vertebrate evolution. *Nature*, 432(7018):695, 2004. ISSN 0028-0836.

Islam, M. S , Thyssen, G. N , Jenkins, J. N , Zeng, L , Delhom, C. D , McCarty, J. C , Deng, D. D , Hinchliffe, D. J , Jones, D. C , and Fang, D. D . A magic population-based genome-wide association study reveals functional association of ghrbb1_a07 gene with superior fiber quality in cotton. *BMC Genomics*, 17(1): 903, 2016. ISSN 1471-2164.

Jain, M , Koren, S , Miga, K. H , Quick, J , Rand, A. C , Sasani, T. A , Tyson, J. R , Beggs, A. D , Dilthey, A. T , and Fiddes, I. T . Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4): 338, 2018. ISSN 1546-1696.

Jannink, J.-L , Lorenz, A. J , and Iwata, H . Genomic selection in plant breeding: From theory to practice. *Briefings in functional genomics*, 9(2):166–177, 2010. ISSN 2041-2649.

Jenko, J , Gorjanc, G , Cleveland, M. A , Varshney, R. K , Whitelaw, C. B. A , Woolliams, J. A , and Hickey, J. M . Potential of promotion of alleles by genome editing to improve quantitative traits in livestock breeding programs. *Genetics Selection Evolution*, 47(1):55, 2015. ISSN 1297-9686.

Ji, L , Neumann, D. A , and Schmitz, R. J . Crop epigenomics: Identifying, unlocking, and harnessing cryptic variation in crop genomes. *Molecular plant*, 8(6): 860–870, 2015. ISSN 1674-2052.

Jiang, Y and Reif, J. C . Modeling epistasis in genomic selection. *Genetics*, 201(2): 759–768, 2015. ISSN 0016-6731.

Jiang, Y , Schmidt, R. H , and Reif, J. C . Haplotype-based genome-wide prediction models exploit local epistatic interactions among markers. *G3: Genes, Genomes, Genetics*, 8(5):1687–1699, 2018. ISSN 2160-1836.

Jiao, Y , Peluso, P , Shi, J , Liang, T , Stitzer, M. C , Wang, B , Campbell, M. S , Stein, J. C , Wei, X , and Chin, C.-S . Improved maize reference genome with single-molecule technologies. *Nature*, 546(7659):524, 2017. ISSN 0028-0836.

Jinek, M , Chylinski, K , Fonfara, I , Hauer, M , Doudna, J. A , and Charpentier, E . A programmable dual-rna–guided dna endonuclease in adaptive bacterial immunity. *Science*, 337(6096):816–821, 2012. ISSN 0036-8075.

Johnsson, M , Gaynor, R. C , Jenko, J , Gorjanc, G , de Koning, D.-J , and Hickey, J. M . Removal of alleles by genome editing (rage) against deleterious load. *Genetics Selection Evolution*, 51(1):14, 2019. ISSN 1297-9686.

Kichaev, G , Roytman, M , Johnson, R , Eskin, E , Lindstroem, S , Kraft, P , and Pasaniuc, B . Improved methods for multi-trait fine mapping of pleiotropic risk loci. *Bioinformatics*, 33(2):248–255, 2017. ISSN 1367-4803.

Kim, S. A and Yoo, Y. J . Effects of single nucleotide polymorphism marker density on haplotype block partition. *Genomics & Informatics*, 14(4):196–204, 2016. ISSN 1598-866X.

Kim, S. A , Cho, C.-S , Kim, S.-R , Bull, S. B , and Yoo, Y. J . A new haplotype block detection method for dense genome sequencing data based on interval graph modeling of clusters of highly correlated snps. *Bioinformatics*, 34(3):388–397, 2017. ISSN 1367-4803.

Klein, R. J , Zeiss, C , Chew, E. Y , Tsai, J.-Y , Sackler, R. S , Haynes, C , Henning, A. K , SanGiovanni, J. P , Mane, S. M , and Mayne, S. T . Complement factor h polymorphism in age-related macular degeneration. *Science*, 308(5720):385–389, 2005. ISSN 0036-8075.

Knott, G. J and Doudna, J. A . Crispr-cas guides the future of genetic engineering. *Science*, 361(6405):866–869, 2018. ISSN 0036-8075.

Kranis, A , Gheyas, A. A , Boschiero, C , Turner, F , Le Yu, Smith, S , Talbot, R , Pirani, A , Brew, F , and Kaiser, P . Development of a high density 600k snp genotyping array for chicken. *BMC Genomics*, 14(1):59, 2013. ISSN 1471-2164.

Kremling, K. A. G , Chen, S.-Y , Su, M.-H , Lepak, N. K , Romay, M. C , Swarts, K. L , Lu, F , Lorant, A , Bradbury, P. J , and Buckler, E. S . Dysregulation of expression correlates with rare-allele burden and fitness loss in maize. *Nature*, 555(7697):520, 2018. ISSN 0028-0836.

La Mara, Casu, S , Carta, A , and Dattena, M . Cryobanking of farm animal gametes and embryos as a means of conserving livestock genetics. *Animal Reproduction Science*, 138(1-2):25–38, 2013. ISSN 0378-4320.

LaFramboise, T . Single nucleotide polymorphism arrays: A decade of biological, computational and technological advances. *Nucleic acids research*, 37(13):4181–4193, 2009. ISSN 0305-1048.

Langmead, B , Trapnell, C , Pop, M , and Salzberg, S. L . Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

Legarra, A , Christensen, O. F , Aguilar, I , and Misztal, I . Single step, a general approach for genomic selection. *Livestock Science*, 166:54–65, 2014. ISSN 1871-1413.

Lewontin, R. C . The interaction of selection and linkage. i. general considerations; heterotic models. *Genetics*, 49(1):49, 1964. ISSN 0016-6731.

Li, N and Stephens, M . Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165(4): 2213–2233, 2003. ISSN 0016-6731.

Li, Z , Simianer, H , and Martini, J. W. R . Integrating gene expression data into genomic prediction. *Frontiers in genetics*, 10:126, 2019. ISSN 1664-8021.

Lin, S , Cutler, D. J , Zwick, M. E , and Chakravarti, A . Haplotype inference in random population samples. *The American Journal of Human Genetics*, 71(5): 1129–1137, 2002. ISSN 0002-9297.

Loh, P.-R , Palamara, P. F , and Price, A. L . Fast and accurate long-range phasing in a uk biobank cohort. *Nature Genetics*, 48(7):811, 2016.

Mackay, T. F. C . The genetic architecture of quantitative traits: Lessons from drosophila. *Current opinion in genetics & development*, 14(3):253–257, 2004.

Malomane, D. K , Simianer, H , Weigend, A , Reimer, C , Schmitt, A. O , and Weigend, S . The synbreed chicken diversity panel: A global resource to assess chicken diversity at high genomic resolution. *BMC Genomics*, 20(1):345, 2019. ISSN 1471-2164.

Manolio, T. A , Collins, F. S , Cox, N. J , Goldstein, D. B , Hindorff, L. A , Hunter, D. J , McCarthy, M. I , Ramos, E. M , Cardon, L. R , and Chakravarti, A . Finding the missing heritability of complex diseases. *Nature*, 461(7265):747–753, 2009. ISSN 0028-0836.

Marchini, J and Howie, B . Genotype imputation for genome-wide association studies. *Nature Reviews Genetics*, 11(7):499, 2010. ISSN 1471-0056.

Marchini, J , Howie, B , Myers, S , McVean, G , and Donnelly, P . A new multipoint method for genome-wide association studies by imputation of genotypes. *Nature Genetics*, 39(7):906, 2007.

Martini, J. W. R , Gao, N , Cardoso, D. F , Wimmer, V , Erbe, M , Cantet, R. J. C , and Simianer, H . Genomic prediction with epistasis models: On the marker-coding-dependent performance of the extended gblup and properties of the categorical epistasis model (ce). *BMC Bioinformatics*, 18(1):3, 2017. ISSN 1471-2105.

Matukumalli, L. K , Lawley, C. T , Schnabel, R. D , Taylor, J. F , Allan, M. F , Heaton, M. P , O'Connell, J , Moore, S. S , Smith, T. P. L , and Sonstegard, T. S . Development and characterization of a high density snp genotyping assay for cattle. *PLOS ONE*, 4(4):e5350, 2009. ISSN 1932-6203.

Mayer, M , Presterl, T , Ouzunova, M , Bauer, E , and Schoen, C. C . Representing allelic diversity of maize landraces by libraries of doubled-haploid lines: paper # 97. *German Plant Breeding Conference, Wernigerode, Germany*, page 138, 2018. URL `http://www.ipk-gatersleben.de/uploads/media/180307_Abstractbuch.pdf`.

McCouch, S. R , McNally, K. L , Wang, W , and Sackville Hamilton, R . Genomics of gene banks: A case study in rice. *American journal of botany*, 99(2):407–423, 2012. ISSN 0002-9122.

McQuillan, R , Leutenegger, A.-L , Abdel-Rahman, R , Franklin, C. S , Pericic, M , Barac-Lauc, L , Smolej-Narancic, N , Janicijevic, B , Polasek, O , and Tenesa, A . Runs of homozygosity in european populations. *The American Journal of Human Genetics*, 83(3):359–372, 2008. ISSN 0002-9297.

Melchinger, A. E , Schopp, P , Müller, D , Schrag, T. A , Bauer, E , Unterseer, S , Homann, L , Schipprack, W , and Schön, C.-C . Safeguarding our genetic resources with libraries of doubled-haploid lines. *Genetics*, 206(3):1611–1619, 2017. ISSN 0016-6731.

Meuwissen, T. H. E . Maximizing the response of selection with a predefined rate of inbreeding. *Journal of animal science*, 75(4):934–940, 1997. ISSN 0021-8812.

Meuwissen, T. H. E , Hayes, B. J , and Goddard, M. E . Prediction of total genetic value using genome-wide dense marker maps. *Genetics*, 157(4):1819–1829, 2001. ISSN 0016-6731.

Meuwissen, T. H. E , Odegard, J , Andersen-Ranberg, I , and Grindflek, E . On the distance of genetic relationships and the accuracy of genomic prediction in pig breeding. *Genetics Selection Evolution*, 46(1):49, 2014. ISSN 1297-9686.

Mezmouk, S and Ross-Ibarra, J . The pattern and distribution of deleterious mutations in maize. *G3: Genes, Genomes, Genetics*, 4(1):163–171, 2014. ISSN 2160-1836.

Moltke, I , Albrechtsen, A , vO Hansen, T , Nielsen, F. C , and Nielsen, R . A method for detecting ibd regions simultaneously in multiple individuals—with applications to disease genetics. *Genome Research*, 21(7):1168–1180, 2011. ISSN 1088-9051.

Money, D , Gardner, K , Migicovsky, Z , Schwaninger, H , Zhong, G.-Y , and Myles, S . Linkimpute: fast and accurate genotype imputation for nonmodel organisms. *G3: Genes, Genomes, Genetics*, 5(11):2383–2390, 2015. ISSN 2160-1836.

Mueller, M. L , Cole, J. B , Sonstegard, T. S , and van Eenennaam, A. L . Comparison of gene editing versus conventional breeding to introgress the polled allele into the us dairy cattle population. *Journal of Dairy Science*, 2019. ISSN 0022-0302.

Nadaraya, E. A . On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

Nakaya, A and Isobe, S. N . Will genomic selection be a practical method for plant breeding? *Annals of botany*, 110(6):1303–1316, 2012. ISSN 1095-8290.

Nei, M . Genetic distance between populations. *The American Naturalist*, 106(949): 283–292, 1972. ISSN 0003-0147.

Nielsen, H. M , Sonesson, A. K , and Meuwissen, T. H . Optimum contribution selection using traditional best linear unbiased prediction and genomic breeding values in aquaculture breeding schemes. *Journal of animal science*, 89(3):630–638, 2011. ISSN 0021-8812.

Nielsen, R . Estimation of population parameters and recombination rates from single nucleotide polymorphisms. *Genetics*, 154(2):931–942, 2000. ISSN 0016-6731.

Pattaro, C , Ruczinski, I , Fallin, D. M , and Parmigiani, G . Haplotype block partitioning as a tool for dimensionality reduction in snp association studies. *BMC Genomics*, 9(1):405, 2008. ISSN 1471-2164.

Patterson, H. D and Thompson, R . Recovery of inter-block information when block sizes are unequal. *Biometrika*, 58(3):545–554, 1971. ISSN 0006-3444.

Pérez, P and de los Campos, G . Genome-wide regression & prediction with the bglr statistical package. *Genetics*, pages 483–495, 2014. ISSN 0016-6731.

Pirani, A , Gao, H , Bellon, L , and Webster, T. A . Best practices for genotyping analysis of plant and animal genomes with affymetrix® axiom® arrays: 2013:p0997, 2013.

Pook, T . Mobps: Modular breeding program simulator: Available at https://github.com/tpook92/mobps; r-package version 1.4.14, 2019a. URL `https://github.com/tpook92/MoBPS`.

Pook, T . Mobpsmaps: Modular breeding program simulator maps: Available at https://github.com/tpook92/mobps; r-package version 0.1.7, 2019b. URL `https://github.com/tpook92/MoBPS`.

Pook, T and Schlather, M . Haploblocker: Creation of haplotype libraries for dhs and highly inbreed lines: Available at https://github.com/tpook92/haploblocker; r-package version 1.4.7, 2019. URL `https://github.com/tpook92/HaploBlocker`.

Pook, T , Weigend, S , and Simianer, H . A generalized approach to calculate expectation and variance of kinship in complex breeding schemes: #26908. *Annual Meeting of EAAP*, 2017a.

Pook, T , Weigend, S , and Simianer, H . Deterministische berechnung von erwartungswert und varianz des inzuchtniveaus in komplexen zuchtprogrammen: # b12. *Vortragstagung der DGfZ und GfT, Hohenheim, Germany*, 2017b.

Pook, T , Schlather, M , de los Campos, G , Schoen, C. C , and Simianer, H . Haploblocker: Creation of subgroup specific haplotype blocks and libraries. *bioRxiv 339788; doi: https://doi.org/10.1101/339788*, 2018.

Pook, T , Schlather, M , de los Campos, G , Mayer, M , Schoen, C. C , and Simianer, H . Haploblocker: Creation of subgroup specific haplotype blocks and libraries. *Genetics*, pages 1045–1061, 2019. ISSN 0016-6731.

Powell, M. J. D . The bobyqa algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pages 26–46, 2009.

Purcell, S , Neale, B , Todd-Brown, K , Thomas, L , Ferreira, M. A. R , Bender, D , Maller, J , Sklar, P , de Bakker, P. I. W , and Daly, M. J . Plink: A tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575, 2007. ISSN 0002-9297.

R Core Team. R: A language and environment for statistical computing, 2017. URL `https://www.R-project.org/`.

Rabbee, N and Speed, T. P . A genotype calling algorithm for affymetrix snp arrays. *Bioinformatics*, 22(1):7–12, 2005. ISSN 1367-4803.

Rabiner, L. R . A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. ISSN 0018-9219.

Ramu, P , Esuma, W , Kawuki, R , Rabbi, I. Y , Egesi, C , Bredeson, J. V , Bart, R. S , Verma, J , Buckler, E. S , and Lu, F . Cassava haplotype map highlights fixation of deleterious mutations during clonal propagation. *Nature Genetics*, 49 (6):959, 2017.

Robinson, M . Bayesian inference of complex trait genetic architecture. *Quantitative Genetics and Genomics Gordon Research Conference*, 2019.

Sabeti, P. C , Reich, D. E , Higgins, J. M , Levine, H. Z. P , Richter, D. J , Schaffner, S. F , Gabriel, S. B , Platko, J. V , Patterson, N. J , and McDonald, G. J . Detecting recent positive selection in the human genome from haplotype structure. *Nature*, 419(6909):832–837, 2002. ISSN 0028-0836.

Sabeti, P. C , Varilly, P , Fry, B , Lohmueller, J , Hostetter, E , Cotsapas, C , Xie, X , Byrne, E. H , McCarroll, S. A , and Gaudet, R . Genome-wide detection and characterization of positive selection in human populations. *Nature*, 449(7164): 913–918, 2007. ISSN 0028-0836.

Sargolzaei, M and Schenkel, F. S . Qmsim: A large-scale genome simulator for livestock. *Bioinformatics*, 25(5):680–681, 2009. ISSN 1367-4803.

Sargolzaei, M , Chesnais, J. P , and Schenkel, F. S . A new approach for efficient genotype imputation using information from relatives. *BMC Genomics*, 15(1): 478, 2014. ISSN 1471-2164.

Schaeffer, L. R . Strategy for applying genome–wide selection in dairy cattle. *Journal of Animal Breeding and Genetics*, 123(4):218–223, 2006. ISSN 1439-0388.

Scheet, P and Stephens, M . A fast and flexible statistical model for large-scale population genotype data: Applications to inferring missing genotypes and haplotypic phase. *The American Journal of Human Genetics*, 78(4):629–644, 2006. ISSN 0002-9297.

Schlather, M , Erbe, M , Skene, F , and Freudenberg, A . miraculix: Algebraic and statistical functions for genetics: Available at https://github.com/tpook92/mobps; r-package version 0.9.7, 2019a. URL `https://github.com/tpook92/MoBPS`.

Schlather, M , Furrer, R , and Kroll, M . Randomfieldsutils: Utilites for the simulation and analysis of random fields: Available at https://github.com/tpook92/mobps; r-package version 0.5.9, 2019b. URL `https://github.com/tpook92/MoBPS`.

Schnable, P. S , Ware, D , Fulton, R. S , Stein, J. C , Wei, F , Pasternak, S , Liang, C , Zhang, J , Fulton, L , and Graves, T. A . The b73 maize genome: complexity, diversity, and dynamics. *Science*, 326(5956):1112–1115, 2009. ISSN 0036-8075.

Schreck, N and Schlather, M . From estimation to prediction of genomic variances: Allowing for linkage disequilibrium and unbiasedness. *bioRxiv 282343; doi: https://doi.org/10.1101/282343*, 2018. URL `https://www.biorxiv.org/content/biorxiv/early/2018/03/15/282343.full.pdf`.

Schrider, D. R , Shanku, A. G , and Kern, A. D . Effects of linked selective sweeps on demographic inference and model selection. *Genetics*, 204(3):1207–1223, 2016. ISSN 0016-6731.

Scott, L. J , Mohlke, K. L , Bonnycastle, L. L , Willer, C. J , Li, Y , Duren, W. L , Erdos, M. R , Stringham, H. M , Chines, P. S , and Jackson, A. U . A genome-wide association study of type 2 diabetes in finns detects multiple susceptibility variants. *Science*, 316(5829):1341–1345, 2007. ISSN 0036-8075.

Shan, Q , Wang, Y , Li, J , and Gao, C . Genome editing in rice and wheat using the crispr/cas system. *Nature protocols*, 9(10):2395, 2014. ISSN 1750-2799.

Siepel, A , Bejerano, G , Pedersen, J. S , Hinrichs, A. S , Hou, M , Rosenbloom, K , Clawson, H , Spieth, J , Hillier, L. W , and Richards, S . Evolutionarily conserved elements in vertebrate, insect, worm, and yeast genomes. *Genome Research*, 15 (8):1034–1050, 2005. ISSN 1088-9051.

Simianer, H . Of cows and cars. *Journal of Animal Breeding and Genetics*, 135(4): 249–250, 2018. ISSN 1439-0388. doi: 10.1111/jbg.12344.

Simianer, H , Pook, T , and Schlather, M . Turning the page - the potential of genome editing in breeding for complex traits revisited. *World Congress on Genetics Applied to Livestock*, page 190, 2018.

Slatkin, M . Linkage disequilibrium—understanding the evolutionary past and mapping the medical future. *Nature Reviews Genetics*, 9(6):477–485, 2008. ISSN 1471-0056.

Sleator, R. D , Shortall, C , and Hill, C . Metagenomics. *Letters in applied microbiology*, 47(5):361–366, 2008. ISSN 0266-8254.

Solberg, L. C , Valdar, W , Gauguier, D , Nunez, G , Taylor, A , Burnett, S , Arboledas-Hita, C , Hernandez-Pliego, P , Davidson, S , and Burns, P . A protocol for high-throughput phenotyping, suitable for quantitative trait analysis in mice. *Mammalian Genome*, 17(2):129–146, 2006. ISSN 0938-8990.

Spratlin, J. L , Serkova, N. J , and Eckhardt, S. G . Clinical applications of metabolomics in oncology: A review. *Clinical cancer research*, 15(2):431–440, 2009. ISSN 1078-0432.

Su, S.-Y , White, J , Balding, D. J , and Coin, L. J. M . Inference of haplotypic phase and missing genotypes in polyploid organisms and variable copy number genomic regions. *BMC Bioinformatics*, 9(1):513, 2008. ISSN 1471-2105.

Sun, X , Stephens, J. C , and Zhao, H . The impact of sample size and marker selection on the study of haplotype structures. *Human Genomics*, 1(3):179, 2004. ISSN 1479-7364.

Swarts, K , Li, H , Romero Navarro, J. A , An, D , Romay, M. C , Hearne, S , Acharya, C , Glaubitz, J. C , Mitchell, S , and Elshire, R. J . Novel methods to optimize genotypic imputation for low-coverage, next-generation sequence data in crop plants. *The Plant Genome*, 7(3), 2014. ISSN 1940-3372.

Taliun, D , Gamper, J , and Pattaro, C . Efficient haplotype block recognition of very long and dense genetic sequences. *BMC Bioinformatics*, 15(1):10, 2014. ISSN 1471-2105.

Täubert, H , Reinhardt, F , and Simianer, H . Zplan+, a new software to evaluate and optimize animal breeding programs. *World Congress on Genetics Applied to Livestock*, page 950, 2010.

Teo, Y. Y . Common statistical issues in genome-wide association studies: A review on power, data quality control, genotype calling and population structure. *Current opinion in lipidology*, 19(2):133–143, 2008. ISSN 0957-9672.

Tyler, A. L , Donahue, L. R , Churchill, G. A , and Carter, G. W . Weak epistasis generally stabilizes phenotypes in a mouse intercross. *PLOS Genetics*, 12(2): e1005805, 2016. ISSN 1553-7404.

Unterseer, S , Bauer, E , Haberer, G , Seidel, M , Knaak, C , Ouzunova, M , Meitinger, T , Strom, T. M , Fries, R , and Pausch, H . A powerful tool for genome analysis in maize: development and evaluation of the high density 600 k snp genotyping array. *BMC Genomics*, 15(1):823, 2014. ISSN 1471-2164.

Unterseer, S , Pophaly, S. D , Peis, R , Westermeier, P , Mayer, M , Seidel, M. A , Haberer, G , Mayer, K. F. X , Ordas, B , and Pausch, H . A comprehensive study of the genomic differentiation between temperate dent and flint maize. *Genome Biology*, 17(1):137, 2016.

Unterseer, S , Seidel, M. A , Bauer, E , Haberer, G , Hochholdinger, F , Opitz, N , Marcon, C , Baruch, K , Spannagl, M , and Mayer, K. F. X . European flint reference sequences complement the maize pan-genome. *bioRxiv*, page 103747, 2017.

VanLiere, J. M and Rosenberg, N. A . Mathematical properties of the r2 measure of linkage disequilibrium. *Theoretical population biology*, 74(1):130–137, 2008. ISSN 0040-5809.

VanRaden, P. M . Efficient methods to compute genomic predictions. *Journal of Dairy Science*, 91(11):4414–4423, 2008. ISSN 0022-0302.

Vogjani, E , Martini, J. W. R , Pook, T , Schoen, C. C , and Simianer, H . Accounting for epistasis improves genomic prediction of phenotypes within and across environments: Poster #202. *Plant 2030 Status Seminar, Potsdam*, 2019, 2019.

Voight, B. F , Kudaravalli, S , Wen, X , and Pritchard, J. K . A map of recent positive selection in the human genome. *PLOS Biology*, 4(3):e72, 2006. ISSN 1545-7885.

Wainschtein, P , Jain, D. P , Yengo, L , Zheng, Z , TOPMed Anthropometry Working Group, Trans-Omics for Precision Medicine Consortium, Cupples, L. A , Shadyab, A. H , McKnight, B , Shoemaker, B. M , Mitchell, B. D , Psaty, B. M , Kooperberg, C , Roden, D , Darbar, D , Arnett, D. K , Regan, E. A , Boerwinkle, E , Rotter, J. I , Allison, M. A , McDonald, M.-L. N , Chung, M. K , Smith, N. L , Ellinor, P. T , Vasan, R. S , Mathias, R. A , Rich, S. S , Heckbert, S. R , Redline, S , Guo, X , Chen, Y.-D. I , Liu, C.-T , de Andrade, M , Yanek, L. R , Albert, C. M , Hernandez, R. D , McGarvey, S. T , North, K. E , Lange, L. A , Weir, B. S , Laurie, C. C , Yang, J , and Visscher, P. M . Recovery of trait heritability from whole genome sequence data. *bioRxiv*, page 588020, 2019. doi: 10.1101/588020. URL `https://www.biorxiv.org/content/biorxiv/early/2019/03/25/588020.full.pdf`.

Wall, J. D and Pritchard, J. K . Haplotype blocks and linkage disequilibrium in the human genome. *Nature Reviews Genetics*, 4(8):587, 2003. ISSN 1471-0056.

Walsh, B and Lynch, M . *Evolution and selection of quantitative traits*. Oxford University Press, 2018. ISBN 0192566644.

Waltz, E . With a free pass, crispr-edited plants reach market in record time, 2018.

Washburn, J. D , Mejia-Guerra, M. K , Ramstein, G , Kremling, K. A , Valluru, R , Buckler, E. S , and Wang, H . Evolutionarily informed deep learning methods for predicting relative transcript abundance from dna sequence. *Proceedings of the National Academy of Sciences*, 116(12):5542–5549, 2019. ISSN 0027-8424.

Weigend, S , Janßen-Tapken, U , Erbe, M , Ober, U , Weigend, A , Preisinger, R , and Simianer, H . Biodiversität beim huhn–potenziale für die praxis. *Züchtungskunde*, 86:25–41, 2014.

Wetterstrand, K . Dna sequencing costs: Data from the nhgri genome sequencing program (gsp), 2019. URL `www.genome.gov/sequencingcostsdata`.

Whalen, A , Ros-Freixedes, R , Wilson, D. L , Gorjanc, G , and Hickey, J. M . Hybrid peeling for fast and accurate calling, phasing, and imputation with sequence data of any coverage in pedigrees. *Genetics Selection Evolution*, 50(1):67, 2018. ISSN 1297-9686.

Willam, A and Simianer, H . *Tierzucht*. UTB, 2017. ISBN 3825248054.

Witten, D. M , Tibshirani, R , and Hastie, T . A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostatistics*, 10(3):515–534, 2009. ISSN 1468-4357.

Yan, G , Qiao, R , Zhang, F , Xin, W , Xiao, S , Huang, T , Zhang, Z , and Huang, L . Imputation-based whole-genome sequence association study rediscovered the missing qtl for lumbar number in sutai pigs. *Scientific Reports*, 7(1):615, 2017. ISSN 2045-2322.

Zerbino, D. R , Achuthan, P , Akanni, W , Amode, M. R , Barrell, D , Bhai, J , Billis, K , Cummins, C , Gall, A , and Girón, C. G . Ensembl 2018. *Nucleic acids research*, 46(D1):D754–D761, 2017. ISSN 0305-1048.

Zhang, K , Calabrese, P , Nordborg, M , and Sun, F . Haplotype block structure and its applications to association studies: Power and study designs. *The American Journal of Human Genetics*, 71(6):1386–1394, 2002. ISSN 0002-9297.

Zhang, P , Zhan, X , Rosenberg, N. A , and Zöllner, S . Genotype imputation reference panel selection using maximal phylogenetic diversity. *Genetics*, 195(2): 319–330, 2013. ISSN 0016-6731.

Zhang, X.-H , Tee, L. Y , Wang, X.-G , Huang, Q.-S , and Yang, S.-H . Off-target effects in crispr/cas9-mediated genome engineering. *Molecular Therapy-Nucleic Acids*, 4:e264, 2015. ISSN 2162-2531.

Zheng, C , Boer, M. P , and van Eeuwijk, F. A . Reconstruction of genome ancestry blocks in multiparental populations. *Genetics*, 200(4):1073–1087, 2015. ISSN 0016-6731.

Zhu, J . Analysis of conditional genetic effects and variance components in developmental genetics. *Genetics*, 141(4):1633–1639, 1995. ISSN 0016-6731.

Zsögön, A , Čermák, T , Naves, E. R , Notini, M. M , Edel, K. H , Weinl, S , Freschi, L , Voytas, D. F , Kudla, J , and Peres, L. E. P . De novo domestication of wild tomato using genome editing. *Nature biotechnology*, 2018. ISSN 1546-1696.

7 List of Figures

2.1 Schematic overview of the steps of the HaploBlocker method: (1) Cluster-building: Classifying local allelic sequences in short windows into groups. (2) Cluster-merging: Simplifying window cluster by merging and neglecting nodes. (3) Block-identification: Identifying blocks based on transition probabilities between nodes. (4) Block-filtering: Creating a haplotype library by reducing the set of blocks to the most relevant ones for the later application. (5) Block-extension: Extending blocks by single windows and SNPs. The same allelic sequences in different steps are coded with the same colors in the graph. 20

2.2 The four parts of the cluster-merging-step. Haplotype frequencies in the window A are according to the toy example given in Table 2.1. 22

2.3 Excerpt of a window cluster. This included all edges (transitions) from the nodes of one of the common paths in the example dataset. 25

2.4 Toy example for the calculation in the block-filtering-step with $w_l = w_n = 1$. 26

2.5 Graphical representation of the block structure for the first 20'000 SNPs of chromosome 1 in the KE DH-lines. Haplotypes are sorted for similarity in SNP 10'000. In that region block structures are most visible and transitions between blocks can be tracked easily. Further away from the centre the representation gets fuzzy. 32

2.6 Proportion of the dataset represented by the haplotype library (coverage) of the training and test set in regard to size of the training set for chromosome 1 in the KE DH-lines. 35

2.7 Estimated and true founders for five representative haplotypes of the last generation of a MAGIC population simulated according to breeding scheme given in (Zheng et al., 2015) using a target coverage of 95% in the generation of the haplotype library. Segments are colored according to the originating/estimated haplotype of the founder generation. 41

2.8 Comparision of EHH and bEHH scores for DH-lines (A) and S_0 (B) for marker 30'000 of chromosome 1 in the KE DH-lines. 41

2.9 IHH scores based on SNPs (A) and haplotype blocks (B) for DH-lines and S_0 for chromosome 1 in the KE. 42

2.10 Comparison of computing times for datasets of various sizes for chromosome 1 in the KE DH-lines. 43

2.11 Dataset for the toy example used in File S3 50

2.12 Comparison of the block structure and an bifurcation plot (Sabeti et al., 2002; Gautier and Vitalis, 2012) according SNP 10'000 on chromosome 1 in the KE DH-lines. 51
2.13 Comparision of the block structure for MCMB=1'000 (A), MCMB=5'000 (B), MCMB=20'000 (C) for the first 20'000 SNPs of chromosome 1 in the KE DH-lines. 52

3.1 LD decay based on physical length (A) and marker distance (B) for chromosome 1 for all considered datasets. Outliers in (A) are corrected for by using a Nadaraya-Watson-estimator (Nadaraya, 1964), using a Gaussian kernel and a bandwidth of 50 kb. (B) is using averaged values for each SNP distance. 57
3.2 Allele specific error rate depending on the allele frequency under different BEAGLE settings for the maize data. Only those dataset entries with the respective allele in the "true" dataset are considered when deriving the allele specific error rate. Y-axis is log-scaled. . . . 61
3.3 Inference error rate based on the location of the genome. Outliers are corrected for by using a Nadaraya-Watson-estimator (Nadaraya, 1964), using a Gaussian kernel and a bandwidth of 3'000 markers for the maize data. Y-axis is log-scaled. 63
3.4 Effect of the parameter *ne* on the inference error rates for the maize data in BEAGLE 5.0 and 5.1. Default settings are indicated by the vertical line. 64
3.5 Effect of the parameter *ne* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 66
3.6 Error rates for UM imputation depending on the size of the reference panel in the maize data. Y-axis is log-scaled. 69
3.7 Error rate per marker for the first 100'000 SNPs according to physical position (starting with chromosome 1) using BEAGLE 5.0 default with B73v4 (Jiao et al., 2017) as a reference genome. 70
3.8 DR2 values in relation to the obtained number of error per marker after fitting of *ne* (A) and on default (B) in BEAGLE 5.0 for the maize data. 50 / 350 DH-lines were used for study / reference sample. 72
3.9 Effect of the inclusion of a single subpopulation in the reference panel based on their genetic distance to the dataset for the chicken diversity panel. Colors according to the subpopulation used as the real dataset in Supplementary Figure 3.11. For a detailed list of subpopulation assignment we refer to Supplementary Table 3.7. Subpopulation 6 (including wild types - turquoise △) is ignored in the regression. . . 73
3.10 Comparison of error rates of UM imputation for different reference panels for the different subpopulations in the chicken diversity panel. Y-axis is log-scaled. For a detailed list on which individual is assigned to which subpopulation we refer to Supplementary Table 3.7. 76

3.11 Neighbor-joining-tree for ten subpopulations in the chicken diversity panel. For a detailed list on which individual is assigned to which subpopulation we refer to Supplementary Table 3.7. 83
3.12 Relationship between region error rate and LD (r^2) on chromosome 9 in the maize data. Outliers are corrected for by using a Nadaraya-Watson-estimator (Nadaraya, 1964), using a Gaussian kernel and a bandwidth of 3'000 markers in both cases. 84
3.13 Total number of errors per marker (50 repetitions) for BEAGLE 4.0 using *buildwindow* of 10 and 1200 (default) in the maize data. 84
3.14 DR2 values in relation to the obtained number of error per marker after fitting of *ne* (A) and on default (B) in BEAGLE 5.0 for the commercial chicken line. 100 / 788 lines were used for study / reference sample. 85
3.15 DR2 values in relation to the obtained number of error per marker after fitting of *ne* (A) and on default (B) in BEAGLE 5.0 for the chicken diversity panel. 100 / 1710 lines were used for study / reference sample. 85
3.16 Effect of the parameter *ne* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 86
3.17 Effect of the parameter *ne* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 86
3.18 Effect of the parameter *err* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 87
3.19 Effect of the parameter *err* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 87
3.20 Effect of the parameter *window* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 88
3.21 Effect of the parameter *window* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 88
3.22 Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 89
3.23 Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 89
3.24 Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 90

3.25 Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 90

3.26 Effect of the parameter *phase-segment* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 91

3.27 Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 91

3.28 Effect of the parameter *phase-stages* on the inference error rates for the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 92

3.29 Effect of the parameter *modelscale* on the inference error rates for the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line. 92

3.30 Effect of the parameter *ne* on the inference error rates for the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line. 93

3.31 Effect of the parameter *buildwindow* on the inference error rates for the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 93

3.32 Effect of the parameter *singlescale* on the inference error rates for the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 94

3.33 Effect of the parameter *nsamples* on the inference error rates for the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 94

3.34 Effect of the parameter *burnin-its* on the inference error rates for the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 95

3.35 Effect of the parameter *ne* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 96

3.36 Effect of the parameter *ne* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 96

3.37 Effect of the parameter *err* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 97

3.38 Effect of the parameter *err* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 97
3.39 Effect of the parameter *window* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 98
3.40 Effect of the parameter *window* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 98
3.41 Effect of the parameter *burnin* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 99
3.42 Effect of the parameter *burnin* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 99
3.43 Effect of the parameter *iterations* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 100
3.44 Effect of the parameter *iterations* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 100
3.45 Effect of the parameter *phase-segment* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 101
3.46 Effect of the parameter *phase-states* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.0. Default settings are indicated by the vertical line. 101
3.47 Effect of the parameter *phase-states* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 5.1. Default settings are indicated by the vertical line. 102
3.48 Effect of the parameter *niterations* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line. 102

3.49 Effect of the parameter *modelscale* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line. 103
3.50 Effect of the parameter *ne* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.1. Default settings are indicated by the vertical line. 103
3.51 Effect of the parameter *buildwindow* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 104
3.52 Effect of the parameter *singlescale* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 104
3.53 Effect of the parameter *nsamples* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 105
3.54 Effect of the parameter *burnin-its* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 105
3.55 Effect of the parameter *phase-its* on the inference and phasing error rates for chromosome 10 of 250 Pseudo S_0 generated based on the maize data in BEAGLE 4.0. Default settings are indicated by the vertical line. 106
3.56 Effect of the parameter *ne* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 107
3.57 Effect of the parameter *err* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 107
3.58 Effect of the parameter *err* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical lines (black for chicken, red for maize). 108
3.59 Effect of the parameter *window* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 108

3.60 Effect of the parameter *window* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 109
3.61 Effect of the parameter *burnin* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 109
3.62 Effect of the parameter *burnin* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 110
3.63 Effect of the parameter *iterations* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 110
3.64 Effect of the parameter *iterations* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 111
3.65 Effect of the parameter *phase-segment* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 111
3.66 Effect of the parameter *phase-states* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 112
3.67 Effect of the parameter *phase-states* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 112
3.68 Effect of the parameter *imp-states* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 113
3.69 Effect of the parameter *imp-states* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 113
3.70 Effect of the parameter *imp-segment* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 114

3.71 Effect of the parameter *imp-segment* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 114
3.72 Effect of the parameter *imp-step* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 115
3.73 Effect of the parameter *imp-nsteps* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 115
3.74 Effect of the parameter *cluster* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.0. Default settings are indicated by the vertical line. 116
3.75 Effect of the parameter *cluster* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 5.1. Default settings are indicated by the vertical line. 116
3.76 Effect of the parameter *ne* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.1. Default settings are indicated by the vertical line. 117
3.77 Effect of the parameter *modelscale* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.1. Default settings are indicated by the vertical line. 117
3.78 Effect of the parameter *niterations* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.1. Default settings are indicated by the vertical line. 118
3.79 Effect of the parameter *buildwindow* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line. 118
3.80 Effect of the parameter *singlescale* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line. 119
3.81 Effect of the parameter *nsamples* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line. 119

3.82 Effect of the parameter *burnin-its* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line. 120
3.83 Effect of the parameter *phase-its* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line. 120
3.84 Effect of the parameter *impute-its* on the UM imputation error rate for the maize data, the commercial chicken line and the chicken diversity panel in BEAGLE 4.0. Default settings are indicated by the vertical line. 121

5.1 Accuracy of genomic prediction on the test set using different datasets to derive the genomic relationship matrix (VanRaden, 2008) for Kemater Landmais Gelb. 133
5.2 Accuracy of genomic prediction using different weightings s for the block length when deriving the genomic relationship matrix for Petkuser Ferdinand Rot. Each line is representing one trait and the red dot is indicating the maximum of the respective curve. 134
5.3 Frequency of genetic material stemming from the wild population after completion of the introgression scheme. The region of the QTL is indicated in red. 137
5.4 Realized kinship when using a cock rotation and a random mating breeding scheme with the same number of chicken. 138
5.5 LD-decay of the base population after 100 generations of random mating via the MoBPS utility function *ld.decay()*. 140
5.6 Plot of the allele frequency in the QTL undergoing a hard sweep via the MoBPS utility function *analyze.population()*. Allele B is favorable and the first mutation is simulated to occur in generation 101. . . . 141
5.7 IHH scores of selection based on EHH (A) and bEHH (B). The location of the sweep is indicated in red. 142
5.8 IHH scores of selection based on EHH (A) and bEHH (B) for an increased mutation rate. The location of the sweep is indicated in red. 142
5.9 One generation of the PAGE selection scheme (reference scenario). 145
5.10 Genetic progress with PAGE (20 edits) vs. the pure GS variant (0 edits) and the proportion of edits on target in the GE generations (inner figure) in the reference scenario. 147
5.11 Extra genetic progress of PAGE vs. GS (left) and proportion of edits on target in generations 11 and 20 (right) for the different scenarios. 147

C.1 Accuracy of genomic prediction on the test set using different genomic datasets to derive the genomic relationship matrix (VanRaden, 2008) for Petkuser Ferdinand Rot. 265

C.2 Accuracy of genomic prediction using different weightings s for the block length when deriving the genomic relationship matrix for Kemater Landmais Gelb. Each line is representing one trait and the red dot is indicating the maximum of the respective curve. 265

8 List of Tables

2.1 Exemplary dataset of allelic sequences and their assignment according to the cluster-building-step. 21
2.2 Influence of MCMB on the haplotype library for chromosome 1 in the KE DH-lines. 33
2.3 Influence of the window size on the haplotype library for chromosome 1 in the KE DH-lines. 34
2.4 Influence of the weighting of block length (w_l) and number of haplotypes (w_n) on the haplotype library for chromosome 1 in the KE DH-lines. 34
2.5 Influence of using the extended-block-identification on the haplotype library in dependency of the parameter t of the extended-block-identification-step for chromosome 1 in the KE DH-lines. 35
2.6 Proportion of variance explained between the full SNP-dataset (X), a SNP-subset (X_s) and the block dataset (Z). For comparability the number of parameters in X_s and Z were chosen equally. 36
2.7 Estimated genomic values using an OLS model assuming additive effects of single markers. 37
2.8 Structure of the haplotype library under different marker densities using the adaptive mode in HaploBlocker with target coverage of 95% for chromosome 1 in the KE DH-lines. 39
2.9 Structure of the haplotype library under different marker densities when adjusting parameters according to data structure for chromosome 1 in the KE DH-lines. 40
2.10 Proportion of variance explained between the full SNP-dataset (X), a SNP-subset (X_s) and the block dataset (Z). For comparability the number of parameters in X_s and Z were chosen equally using traditionally heritability estimation as in (de los Campos, 2017). . . 48
2.11 Influence of the window size on the haplotype library for chromosome 1 in the KE DH-lines with a target coverage of 95%. 48
2.12 Influence of the weighting of block length (w_l) and number of haplotypes (w_n) on the haplotype library for chromosome 1 in the KE DH-lines with a target coverage of 95%. 49
2.13 Influence of using the extended-block-identification on the haplotype library in dependency of the parameter t of the extended-block-identification-step for chromosome 1 in the KE DH-lines with a target coverage of 95%. 49

3.1 Inference error for the KE DH-lines by changing a single imputing parameter. . . . 62

3.2 Inference and phasing error for the 250 Pseudo S_0 lines based on the KE DH-lines for chromosome 10. * BEAGLE crashed for this dataset when using *phase-segment* $>$ 10, *phase-states* $<$ 100 or *phase-states* $>$ 10'000. . . . 65

3.3 UM imputation error for the KE DH-lines by changing a single imputing parameter with ne = 1'000 for BEAGLE 5.0 / 5.1 and ne = 300 for BEAGLE 4.1. . . . 67

3.4 Inference error rates using different reference genomes compared to B73 for KE DH-lines. Only markers mapped on both the flint reference genome & B73v4 (Jiao et al., 2017) are considered for "critical" markers (error rate $>$ 10%). . . . 71

3.5 List of "critical" markers for Kemater Landmais Gelb using different reference genomes. . . . 77

3.6 List of "critical" markers for Petkuser Ferdinand Rot using different reference genomes. . . . 77

3.7 Assignments to subpopulations for the chicken diversity panel based on Nei standard genetic distances (Nei, 1972). . . . 77

3.8 UM imputation error for the commercial breeding line in chicken by changing a single imputing parameter with ne = 100 for BEAGLE 5.1, ne = 300 for BEAGLE 5.0 and ne = 10'000 for BEAGLE 4.1. * BEAGLE crashed for this dataset when using *phase-segment* $>$ 10, *phase-states* $<$ 100 or *imp-step* $>$ 5. . . . 78

3.9 UM imputation error for the diversity panel in chicken by changing a single imputing parameter with ne = 300 for BEAGLE 5.1, ne = 3'000 for BEAGLE 5.0 and ne = 10'000 for BEAGLE 4.1. * BEAGLE crashed for this dataset when using *phase-segment* $>$ 10 or *phase-states* $<$ 100. . . . 79

3.10 Phasing error, as number of heterozygous markers per switch error, for Pseudo S_0 generated based on the KE DH-lines by changing a single imputing parameter. . . . 80

3.11 Inference error rates using different reference genomes compared to B73 for PE DH-lines. Only markers mapped on both the flint reference genome & B73v4 (Jiao et al., 2017) are considered for "critical" markers (error rate $>$ 10%). . . . 81

3.12 Error rates for UM imputation for different reference panels, including A (same subpopulation), C (all other subpopulation), D (below average Nei distant subpopulations), E (All subpopulation with reduced error rate when testing A + B compared to A as the reference panel). . . . 81

3.13 Minimal obtained error rates and used parameter settings for inference and UM imputation. Deviations from the ideal single parameter settings are caused by BEAGLE crashing when changing parameters jointly. 82

A Guidelines to HaploBlocker

CiBreed
Center for Integrated Breeding Research

HAPLOBLOCKER

Torsten Pook, Martin Schlather

5. MAI 2019

UNIVERSITY OF GOETTINGEN
Animal Breeding and Genetics Group

Table of Contents

1 Preface 2

2 Installation 2

3 Citation 2

4 General 3

5 Parameters of the main function (block_calculation) 3

5.1 Prefilters 3

5.2 Cluster-building 4

5.3 Cluster-merging 4

5.4 Block-identification 5

5.5 Block-filtering 5

5.6 Block-extending 6

5.7 Off-variant-identification (optional) 6

5.8 Performance parameters – computing time: 6

6 Exemplary inputs 7

7 Output 9

8 Data Availability 10

9 Functions for later analysis 10

9.1 plot_block() 10

9.2 blocklist_startend() 11

9.3 coverage_test() 11

9.4 block_matrix_construction() 12

9.5 block_windowdataset() 12

9.6 block_ehh() 13

9.7 block_ihh() 13

9.8 block_plot() 14

9.9 blocklist_plot() 14

9.10 blocklist_plot_xsize() 15

10 Acknowledgements 16

1 Preface

HaploBlocker is an R-package to compute a haplotype block library according to our paper "HaploBlocker: Creation of subgroup specific haplotype blocks and libraries". The publication is currently available on *biorvix* (https://www.biorxiv.org/content/10.1101/339788v2) and submitted to *Genetics*. In the following, we will give some short guidelines on how to use the package and introduce input parameters to change the structure of the resulting haplotype library.

2 Installation

HaploBlocker requires R 3.0+ (and the included graphics and stats package) as well as the R-package RandomFieldsUtils (version 0.4.0+). RandomFieldsUtils is available on CRAN and at https://github.com/tpook92/HaploBlocker. HaploBlocker can directly be installed via the function install_github from the package devtools:

Devtools::install_github(tpook92/HaploBlocker", subdir="pkg")

For manual installation of the package, use the R usage the R function *install.packages* (under windows set type="source", repo=NULL). Usage was tested on Linux and Windows. The usage on Mac OS is currently not recommended.

For Windows the installation of Rtools is required. Some machines additionally require devtools.

3 Citation

There is currently no published version of our manuscript in a peer-reviewed journal. This will hopefully change soon. For so long we suggest using following the citations for the preprint on *biorvix* and the R-packages HaploBlocker:

```
@article{Pook.2018,
 author = {Pook, Torsten and Schlather, Martin and {de los Campos}, Gustavo and Mayer, Manfred and Schoen, Chris Carolin and Simianer, Henner},
 year = {2019},
 title = {HaploBlocker: Creation of subgroup specific haplotype blocks and libraries},
 doi = {10.1101/339788}
 journal = {bioRxiv}
}

@misc{Pook.2018b,
 author = {Pook, Torsten and Schlather, Martin},
 year = {2018},
 title = {HaploBlocker: An R package for the Creation of Haplotype Libraries for DHs and Highly Inbreed Lines},
 url = {https://github.com/tpook92/HaploBlocker}
}
```

4 General

The main function of HaploBlocker is *block_calculation()* and the only mandatory input of the function is a dataset (parameter: **dhm**) containing haplotypes or a path to a vcf/ped-file to import – inputs can contain up to 256 different characters/numeric/integer values. For maximum internal efficiency use just two variants:

```
> ex_maze[1:10,1:8]
       Haplo 1 Haplo 2 Haplo 3 Haplo 4 Haplo 5 Haplo 6 Haplo 7 Haplo 8
SNP 1        T       C       T       C       T       C       T       C
SNP 2        A       A       A       A       A       A       A       A
SNP 3        C       C       C       C       C       C       C       C
SNP 4        A       A       A       A       A       A       A       A
SNP 5        T       T       T       T       G       T       T       T
SNP 6        C       C       C       C       T       C       C       C
SNP 7        G       G       G       G       G       G       G       G
SNP 8        G       G       G       G       G       G       G       G
SNP 9        A       C       A       C       A       C       A       C
SNP 10       C       C       C       C       C       C       C       C
```

Figure 1: Excerpt of the dataset ex_maze

When running, the user is receives updates regarding the currently stage of the algorithm:

```
> blocklist <- block_calculation(ex_maze)
Start_blockinfo_calculation
.....................Start_nodes_calculation
.....................Start_simple_merge: 2073
Start_CrossMerging_full
Iteration 1 : 1100 nodes
Iteration 2 : 799 nodes
Iteration 3 : 773 nodes
Iteration 4 : 766 nodes
Start_IgnoreSmall
Iteration 1 : 766 nodes
Iteration 2 : 588 nodes
Iteration 3 : 586 nodes
Iteration 4 : 585 nodes
Start_Blockmerging
Iteration 1 : 745 blocks
Iteration 2 : 436 blocks
Iteration 3 : 112 blocks
Iteration 4 : 86 blocks
Iteration 5 : 70 blocks
Start_Blockextending
Iteration 1 : 70 blocks;  0 block extensions
Iteration 2 : 70 blocks;  9 block extensions
```

Figure 2: Example usage of the function block_calculation in Haploblocker

5 Parameters of the main function (block_calculation)

In the following, we will discuss the parameters for tuning of the structure of the derived haplotype library according to the step they are occurring in the algorithm. For exemplary inputs, we refer to section 6. As the default settings are chosen to work for most dataset but still perform fast we recommend the usage of our adaptive mode (**adaptive_mode**=TRUE) to create a haplotype library without modifying parameters. The interested reader is referred to the manuscript for a more detailed discussion and comparison of the structure of different haplotype libraries obtained under different parameter settings.

5.1 Prefilters

Parameters: prefilter, maf, equal_remove

Before the actual algorithm is executed, one can remove non-informative SNPs of the dataset. Firstly, filtering can be performed according to a minor allele frequency filter (parameter **maf**). Secondly, one can remove all SNPs in perfect LD to the previous one by activating **equal_remove**. On default, no filtering is done and it has to be activated via the parameter **prefilter**.

5.2 Cluster-building

Parameters: window_sequence, window_size, merging_error, max_groups, bp_map, window_anchor_gens, blockinfo_mode, at_least_one, multi_window_mode, blockinfo_mode_na, na_snp_weight, na_seq_weight, actual_snp_weight

On default, the windows of the dataset are of equal length (**window_size**) and in every window the same number of errors (**merging_error**) is allowed. In case one wants to use different window sizes and number of errors per region (e.g. to exactly span windows according to the position of genes) one can manual set this up via the parameter **window_sequence**. To include the position in base pairs one has to enter the position of each SNP via the parameter **bp_map**.

Since the manual input can be tiring, we offer an additional possibility to generate a **window_sequence**. The parameter **max_groups** can be used to chose the window boundaries to obtain a certain number of variants per window (Next window starts whenever the previous block would have more variants than **max_groups**).

When providing the physical position of wanted window boundaries (e.g. start/end point of genes) provide them in the parameter **window_anchor_gens** and the resulting **window_sequence** is automatically calculated. Note that no overlapping windows are supported!

To minimize the number of groups in each window one can use the parameter **blockinfo_mode** (on default the groups are derived according to the most common haplotypes in the window). **At_least_one** is an utility parameter to make sure that in each window at least one SNP has to be the same (only relevant for **window_size ≤ merging_error**)

To use multiple window clusters in the fitting procedure set **multi_window_mode** to TRUE. By doing this **window_size**, **merging_error** and **min_share** are able to process vectors as input and/or **window_sequence** can be just as a list of different window sequences. Each element is processed separately. In case no vector/list is provided the input is used for all cases.

In case the dataset contains missing values, those on default will be modelled as another allelic variant (“9”) in the analysis. To count differences between NAs and allelic variants with different weighting activate **block_mode_na**. The difference between NA and an allelic variant is counted as **na_snp_weight** merging errors whereas different allelic variants are counted as **actual_snp_weight** merging errors. In case a marker contains only one allelic variant and NAs differences are counted as **na_sep_weight** merging errors. It has to be noted there that this mode is significantly more time consuming and still open to some change. Note that the required input of our tool are haplotypes – so phasing is required before application.

5.3 Cluster-merging

Parameters: node_min, gap, min_reduction_cross, min_reduction_neglet, early_remove, node_min_early

In the cluster-merging the number of haplotypes per node can be controlled via **node_min**. To avoid short segments between removed nodes, all haplotypes are in a common variant for less than **gap** windows are removed from the window cluster.

To reduce computing time in the SG,SM and NN,SG,SM, SG cycles one can use **min_reduction_cross** and **min_reduction_neglet** to stop the cycles when there are less than that many merges occuring in one cycle of the algorithm. In case there is a high number of nodes with few haplotypes in it, one can consider removing them before the SG, SM cycle via **early_remove** and **node_min_early**.

5.4 Block-identification

Parameters: min_share, subgroups, consider_nodes, consider_edge, min_per_subgroup, consider_multi, multi_min, node_min, edge_min, double_share

To not consider nodes or edges as starting blocks in the identification step one can set the **consider_nodes**, **consider_edge** to FALSE. Both those changes are not recommended (setting one to FALSE will decrease computing time). To additionally screen for blocks based on haplotypes in two adjacent edges use **consider_multi** (only recommened for small dataset & use **multi_window_mode** first). To change the minimum number of haplotypes per block one can use **edge_min** (Blocks by Edge), **node_min** (Blocks by node) and **multi_min** (multiple edges).

To change the minimum proportion of a block transitioning in the same node required to extend the block one can use the parameter **min_share**. By doing this one can control the average length of each block and the similarity between haplotypes from a block. A higher value leads to shorter blocks and therefore leads to a higher similarity between the haplotypes in a block and a higher number of blocks overall. Additionally, the number of overlapping blocks is heavily reduced.

To form blocks not only for the whole dataset but also for subgroups one can use the parameter **subgroups** and set the minimum number haplotypes of each subgroup to be in each block (**min_per_subgroup**). A change in this parameter leads to blocks which are in all subgroups of the dataset and therefore can lead to low coverages. A change here is only recommended when one is explicitly interested in overlapping regions of multiple subgroups.

To consider both the long and the short segments in the block-identification (extended-block-identification) set **double_share** to the minimum share of the haplotypes required to transition in the same longer segment.

5.5 Block-filtering

Parameters: min_majorblock, min_majorblock_steps, min_similarity, save_allblock, consider_all, merge_closeblock, max_diff_i, max_diff_l, off_lines, weighting_length, weighting_size, target_coverage, target_stop

The main filtering process is performed by identifying the number of cells in which each block is the most relevant block of the dataset. This number can be changed via **min_majorblock** and should be used to find a balance between the number of blocks and the coverage of the block library. To obtain a haplotype library with a specific coverage we recommend the use of the parameter **target_coverage** to initialize an automatic fitting procedure to determine a good choice for **min_majorblock**. To control the number of iterations performed to fit **min_majorblock** in **target_coverage** use **max_iteration** with

min_step_size controlling the minimal difference in **min_majorblock** per step and **target_stop** providing a maximum difference to the target.

To control which block is the most relevant block in each cell one can control the weighting between the length and number of haplotypes in each block by using the parameters **weighting_length** and **weighting_size**.

To avoid excluding important blocks, the minimum number is increased slowly (in **min_majorblock_steps** linear increasing steps). The minimum similarity of a haplotype with a block to be included can be set via the parameter **min_similarity**. By doing this, one can control the minimum similarity between two haplotypes of the same block. Haplotypes not fulfilling **min_similarity** but being in all node used to identify the block are not removed unless the parameter **save_allblock** is set to FALSE.

Additionally, there are some minor parameters in the filtering process. To not consider haplotypes which are not in the block one has to set **consider_all** to FALSE. To allow blocks with similar haplotypes and location to be merged one has to activate **merge_closeblock** and set the maximum differences between them via **max_diff_i** (different haplotypes) and **max_diff_l** (differences between both). The minimum number of additional haplotypes a block has to have compared to another block when the sequence of windows is the same can be controlled via **off_lines**.

5.6 Block-extending

Parameters: block_extending, max_extending_diff, extending_ratio, snp_extending, max_extending_diff_snp, extending_ratio_snp

If one does not want the block and SNP extension to be performed set **block_extending** and/or **snp_extending** to FALSE. If one wants to use it one can control the maximum number of windows that are different in some haplotypes (**max_extending_diff**, **max_extending_diff_snp**) and ratio between windows with and without variation (**extending_ratio, extending_ratio_snp**).

5.7 Off-variant-identification (optional)

This step is not executed on default as its application is only recommend to obtain an absolute maximum coverage for a dataset. Here, in addition to the window cluster additional blocks are generated based on those entries/cells of the dataset not included in the block library before.

Parameters: off_node_addition, raster, off_node_minimum_blocklength, off_node_minimum_size

Set **off_node_addition** to TRUE to active this step. In this step each cell is screen for a section of **off_node_minimum_size** haplotypes with the same sequence in **off_node_minimum_blocklength** windows. Afterward all other steps are executed again (especially filtering for **min_majorblock**).

To reduce computing time not every window is consider, but instead only each **raster** window (this should be a value smaller than **off_node_minimum_blocklength**).

5.8 Performance parameters – computing time:

Parameters: recoding, recoding_notneeded, fast_compiler, intersect_func, c_dhm_mode, parallel_window, window_overlap, window_cores

Internal computations are faster when a low number of different characters is use is the input dataset (**dhm**). To change the coding to major_variant "A", minor_variant "C" in every SNP set the parameter **recoding** to TRUE. If this is already done you can further use **recoding_notneeded** to skip the recoding step and still profit from the advantages of the recoding.

To further reduce computing time one can active parallel computing via **parallel_window**. Here the dataset is split into windows containing **parallel_window** markers. On defaults, window do not overlap but in principle can via **window_overlap**. The number of cores used is controlled via **window_cores**.

Fast_compiler enables the compiler-packages and just-in-time computing. **Intersect_func** loads in a more efficient variant of base::intersect and **c_dhm_mode** controls if bitwise coding is used internally (don't see a reason why you would not want this).

6 Exemplary inputs

Parameter-name	Default	Other option:
prefilter	FALSE	TRUE
maf	0.00	Value between 0 and 0.5
equal_remove	FALSE	TRUE

window_sequence	NULL (automatically generated)	`> window_sequence[1:5,]`
		`Start-SNP End-SNP Length Min-Same Start-BP End-BP`
		`[1,] 1 20 20 19 0 0`
		`[2,] 21 40 20 19 0 0`
		`[3,] 41 60 20 19 0 0`
		`[4,] 61 80 20 19 0 0`
		`[5,] 81 100 20 19 0 0`

window_size	20	Natural number (1,2,3,…)
merging_error	1	Natural number (1,2,3,…) - lower than window_size!
max_groups	0	to active: Natural number >= 2
bp_map	NULL	`base_pair` `[1] 48208 49308 49527 50846 51053 52778`

window_anchor_gens	NULL	`BP-start BP-end`
		`GRMZM2G354611 66347 68582`
		`GRMZM2G100965 169103 170546`
		`GRMZM5G833275 176866 177244`
		`GRMZM2G100979 183185 183884`
		`GRMZM2G310569 311374 318659`
		`GRMZM2G341658 563756 566860`
		`GRMZM2G039325 567358 580418`
		`GRMZM2G415022 706080 707213`
		`AC195959.2_FG004 709107 712584`
		`GRMZM2G307823 725998 729956`

blockinfo_mode	0	1 to minimize groups per window
at_least_one	TRUE	FALSE
multi_window_mode	FALSE	TRUE (use e.g. window_size=c(5,10,20,50))
blockinfo_mode_na	FALSE	TRUE (adjust merging_error !)
na_snp_weight	2	Numeric value >0
na_seq_weight	0	Numeric value > 0
actual_snp_weight	5	Numeric value > 0
gap	10	Natural number (1,2,3,…)
min_share	0.975	Value between 0.5 and 1 (highly recommend to not use small values!

node_min	5	Natural number (1,2,3,…)
edge_min	5	Natural number (1,2,3,…)
multi_min	5	Natural number (1,2,3,…)
consider_nodes	TRUE	FALSE
consider_edge	TRUE	FALSE
consider_multi	FALSE	TRUE
subgroups	NULL (automatic generated)	List(1:500, 1:200, 1:300) Subpopulation 1 in first 200 colums Subpopulation 2 in last 300 colums
min_per_subgroup	0	Natural number (1,2,3,…) Only when one is explicitly interested in the overlap between both populations!
min_majorblock	5'000	Non-negativ-number (0,1,2,…)
min_majorblock_steps	4	Non-negativ-number (0,1,2,…)
min_similarity	0.99	Value between 0 and 1 (highly recommend to not use values below 0.9!
save_allblock	TRUE	FALSE
consider_all	TRUE	FALSE
merge_closeblock	FALSE	TRUE
max_diff_i	1	Non-negative-number (0,2,3,…)
max_diff_l	1	Non-negative-number (0,2,3,…)
off_lines	2	Natural number (1,3,4m…)
weighting_length	1	Numeric value (<0 not recommended)
weighting_size	1	Numeric value (<0 not recommended)
block_extending	TRUE	FALSE
snp_extending	TRUE	FALSE
max_extending_diff	1	Non-negative-number (0,2,3,…)
max_extending_diff_snp	0	Non-negative-number (1,2,3,…)
extending_ratio	20	Natural number (1,2,3,…) Avoid low values
extending_ratio_snp	Inf	Naturual number (1,2,3,…) Only change for long windows and high number of haplotypes in blocks
off_node_addition	FALSE	TRUE
raster	5	Natural number (1,2,3,…)
recoding	FALSE	TRUE
recoding_notneeded	FALSE	TRUE
fast_compiler	TRUE	FALSE
intersect_func	TRUE	FALSE will use base::intersect HaploBlocker::intersect requires that vector in an ascending sequence of numerics

c_dhm_mode	TRUE	FALSE
big_output	FALSE	TRUE
target_coverage	NULL	Value between 0 and 1
max_iteration	10	Natural number (1,2,3,…)
min_step_size	25	Natural number (1,2,3,…)
target_stop	0.001	Value between 0 and 1 (recommend close to 0)
multi_window_mode	FALSE	TRUE; Can be actived by using a vector for window_size; merging_error or min_share
adaptive_mode	FALSE	TRUE; Sets window_size = c(5,10,20,50) and Target_coverage = 0.9
developer_mode	FALSE	TRUE
parallel_window	Inf	Natural number – bigger than the biggest blocks one wants to identify
window_overlap	0	Natural number – nothing bigger than the size of the largest block is needed
window_cores	1	Natural number (2,3,4,…)
double_share	1	Value between 0 and 1 (nothing below 0.5 is recommended)
min_reduction_cross	-Inf	Non-negative-number (0,1,2,3,…)
min_reduction_neglet	-Inf	Non-negative-number (0,1,2,3,…)
early_remove	FALSE	TRUE
node_min_early	NULL	Natural number (1,2,3,…) – e.g. node_min / edge_min

7 Output

The output of *block_calculation()* is a list containing a block in each element. For each block the following information are stored:

1. Sequence of nodes in the window cluster
2. Start of the block (in windows, SNPs and bp)
3. End of the block (in windows, SNPs and bp)
4. Sequence of the group in each window
5. Number of haplotypes in the block
6. List of Haplotypes in the block
7. 1. Sequence of alleles in the block (joint allelic sequence)
7. 2. Frequency of the joint allelic sequence per marker
8. – 12. Internal stuff to save computing time in the algorithm (Only in the output using developer-mode)

To not only generate the haplotype library but additionally the window-dataset, the window-cluster and general information on each window get the parameter **big_output** to TRUE.

8 Data Availability

A full dataset containing 80'200 markers for 910 individuals is provided with the publication and is included in our GitHub repository (https://github.com/tpook92/HaploBlocker). All results presented in the publication are limited to chromosome 1. Datasets for other chromosomes will be made available with publication of other project partners and hopefully then included in the package. The package itself contains a dataset of the first 9'999 SNPs of 313 KE DH-lines (**ex_maze**).

9 Functions for later analysis

So far, we have only implemented smaller utility functions to assess the relevant parts of the output and generate basic plots to get an overview of the structure of the blocks. We are always happy for feedback on additional wishes for possible outputs or other options to include in our algorithm.

Usage of function 9.1, 9.2, 9.3 is encouraged to get a general feeling about the structure of the haplotype library. Both 9.4 and 9.5 can be used the generated block datasets for later analysis. 9.6 and 9.7 contain function to derive bEHH and iHH scores for the haplotype library. 9.8, 9.9, 9.10 are old utility functions to get a generate feeling about the structure of the haplotype library.

9.1 plot_block()

Parameter: blocklist, type="snp", orientation="snp", include=TRUE, indi=NULL, min_to_plot=5, intensity=0.5, add_sort=TRUE, max_step=500, snp_ori=NULL, export_order=FALSE, import_order=FALSE

This function can be used to generate a graphical representation of a **blocklist**. Use **type** to select scaling of the x-axis ("bp", "snp", "window"). To sort haplotypes use the parameter **orientation** – To align against blocks in set **orientation** to "front", "mid" or "back". We recommend aligning against a location in SNPs. On default the middle of the dataset is used but can be manually set using **snp_ori**. For ordering haplotypes only the adjacent **max_step** blocks are considered – the default of 500 should be more than enough for all applications. Instead of using our sorting algorithm one can import the order of haplotypes using **import_order** (or export using **export_order**=TRUE).

Only those blocks are displayed with at least **min_to_plot** haplotypes in it. To show overlap, blocks are displayed with a low color **intensity**.

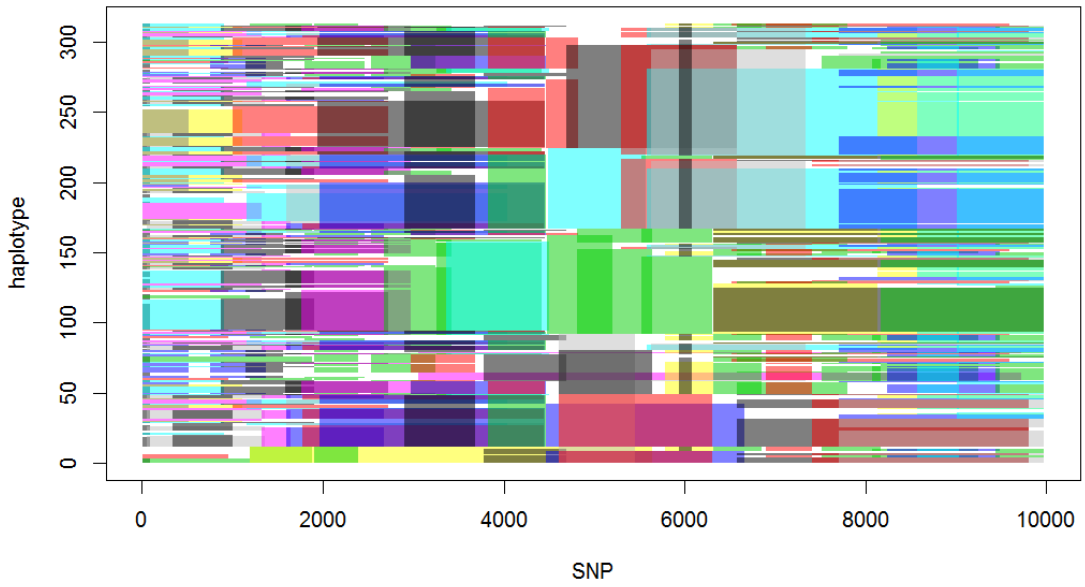


Figure 3: Exemplary output of plot_block for the ex_maze blocklist.

9.2 blocklist_startend()

Parameters: blocklist, type="snp", first_block=1

Calculate the start and endpoint of each block. Select the **type** ("window", "snp", "bp") accordingly. Use **first_block** to skip deriving start/endpoints of the first blocks (this is needed internally).

```
> blocklist_startend(blocklist)
         start  end
block 1      1   89
block 2      1  955
block 3      1 1192
block 4      7  521
block 5      7  903
block 6      7 1325
block 7      7 1112
block 8     54 1365
block 9    340 1003
block 10   749 1367
block 11   869 1902
block 12  1000 2723
block 13  1149 1556
block 14  1155 4041
block 15  1194 1883
```

Figure 4: Exemplary output of blocklist_startend for the ex_maze blocklist.

9.3 coverage_test()

Parameters: blocklist, indi=NULL, type="snp", max=1

Calculate which cells of the dataset are covered by any block. Set **max** to a value above 1 to display how many blocks are presented in each cell. For big datasets setting **type** to "window" is recommended to reduce computing time.

```
> t <- coverage_test(blocklist)
> mean(t)
[1] 0.9097332
```

9.4 block_matrix_construction()

Parameter: blocklist

Calculate a block-dataset according to the block library

```
> new_data <- block_matrix_construction(blocklist)
> new_data[1:8,1:8]
       haplo1 haplo2 haplo3 haplo4 haplo5 haplo6 haplo7 haplo8
block1      1      0      1      0      0      0      1      0
block2      0      0      0      0      0      0      0      0
block3      0      0      0      0      0      0      0      0
block4      0      1      0      1      0      1      0      1
block5      0      0      1      0      1      0      0      0
block6      0      0      0      0      0      0      0      0
block7      0      1      0      1      0      1      0      0
block8      1      0      0      0      0      0      0      0
```

Figure 5: Exemplary output of block_matrix_construction for the ex_maze blocklist

9.5 block_windowdataset()

Parameter: blocklist=NULL, data=NULL, consider_nonblock=FALSE, return_dataset=FALSE, non_haploblocker=FALSE

Generate a window-based block dataset. Blocks span over the same windows of markers for better comparability to other block based approaches. Overall, windows are much shorter than HaploBlocker blocks. Set **consider_nonblock** to TRUE to haplotypes in no haplotype block in HaploBlocker to be in blocks. Set **return_dataset** to TRUE to instead of a dataset coding presence/absence allow for more variants in each window.

To ignore the haplotype blocks derived and just compute all variants set **non_haploblocker** to TRUE. In case **consider_nonblock** or **non_haploblocker** are used, the original dataset has to be provided via **data**.

```
> new_data <- block_windowdataset(blocklist)
> new_data[1:8,1:8]
                    haplo1 haplo2 haplo3 haplo4 haplo5 haplo6 haplo7 haplo8
window:1-6variant1       1      0      1      0      0      0      1      0
window:1-6variant2       0      0      0      0      0      0      0      0
window:7-53variant1      1      0      1      0      1      0      1      0
window:7-53variant2      0      0      0      0      0      0      0      0
window:7-53variant3      0      1      0      1      0      1      0      1
window:7-53variant4      0      0      0      0      0      0      0      0
window:54-89variant1     1      0      1      0      1      0      1      0
window:54-89variant2     0      0      0      0      0      0      0      0
```

Figure 6: Exemplary output of block_windowdataset for the ex_maze blocklist

```
> new_data <- block_windowdataset(blocklist, data= ex_maze, non_haploblocker = TRUE)
Start_blockinfo_calculation
.....................
> new_data[1:8,1:8]
                    haplo1 haplo2 haplo3 haplo4 haplo5 haplo6 haplo7 haplo8
window:1-6variant1       1      0      1      0      0      0      1      0
window:1-6variant2       0      1      0      1      0      1      0      1
window:1-6variant3       0      0      0      0      1      0      0      0
window:1-6variant4       0      0      0      0      0      0      0      0
window:1-6variant5       0      0      0      0      0      0      0      0
window:1-6variant6       0      0      0      0      0      0      0      0
window:1-6variant7       0      0      0      0      0      0      0      0
window:7-53variant1      1      0      1      0      0      0      1      0
```

Figure 7: Exemplary output of block_windowdataset for the ex_maze blocklist considering all variants.

9.6 block_ehh()

Parameters: blocklist, data=NULL, marker, plot=FALSE, position1=NULL, standardization=3, group=NULL, return_ehh=TRUE

This function can be used to derive bEHH scores for a given **blocklist**. In case no blocklist is provided a SNP-dataset can be provided in **data**. bEHH is computed the marker given in **marker**. To plot the bEHH curve set **plot** to TRUE. On default, distance between markers are assumed to be equidistant. Position of markers can be provided via **position1**.

A change of **standardization** is not recommended for external use. To compute bEHH scores for different subgroups use **group**. To instead of bEHH return iHH set **return_ehh** to FALSE.

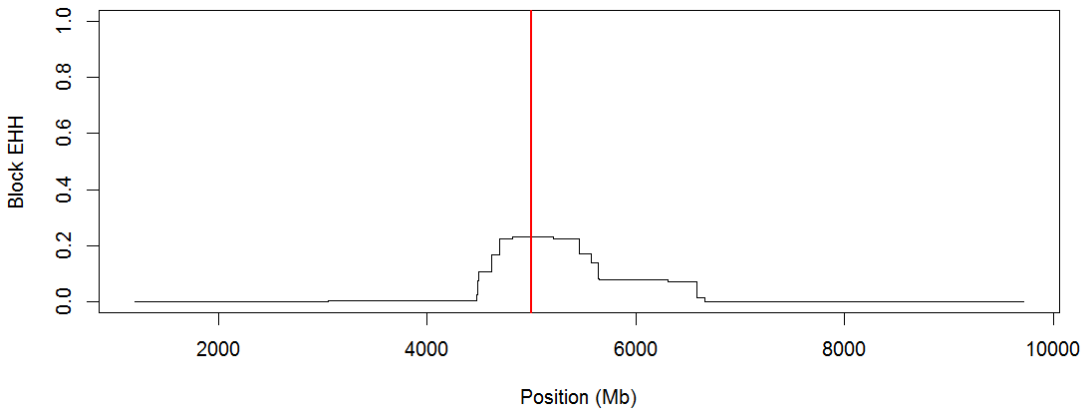


Figure 8: Exemplary output of block_ehh (marker=5000, plot=TRUE).

9.7 block_ihh()

Parameters: blocklist, data=NULL, plot=FALSE, position1=NULL, standardization=3, group=NULL

This function can be used to compute iHH scores for the whole genome – input parameters work in the same way as *block_ehh* (9.6).

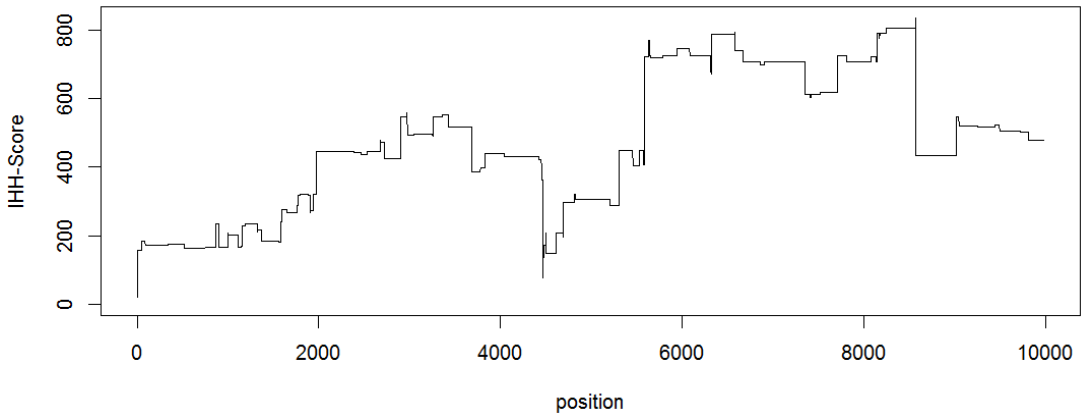


Figure 9: Exemplary output of block_ihh (plot=TRUE).

9.8 block_plot()

Parameters: blocklist, indi=NULL, type="snp", bw=1

This function can be used to generate a plot of each blocks position (x-axis length, y-axis number of haplotypes in block). The red lines indicated the coverage of the full block library per region.

The parameter **indi** is automatically calculated. **Type** can be set to "window", "snp" or "bp" depending on the desired scaling. The use of "window" is not recommended when using multiple window sizes. For smoothing of the coverage curve a Nadaraya-Watson estimator with bandwidth **bw** can be used (default: no smoothing).

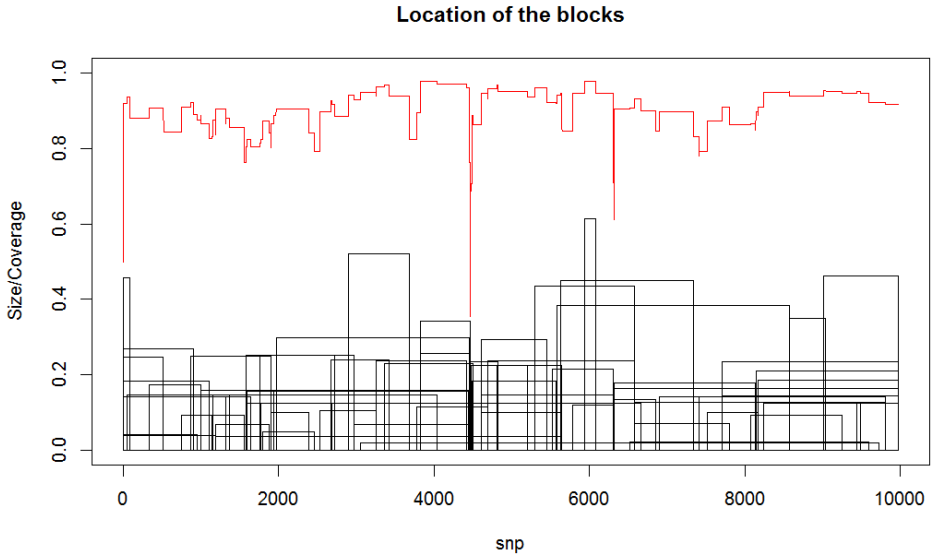


Figure 10: Exemplary output of block_plot for the ex_maze blocklist

9.9 blocklist_plot()

Parameters: blocklist, cutoff2=5, bound_weighted=TRUE, type="snp"

The location of the blocks is given according to the y-axis. Additionally, recombination hotspots are indicated by horizontal lines. **Cutoff2** (for the example 3 was used) is the minimum number of blocks that are required in that region to mark a position as a hotspot and **bound_weighted** scales blocks according to the size of the block. We do not only count the position itself but adjacent markers via a kernel regression method.

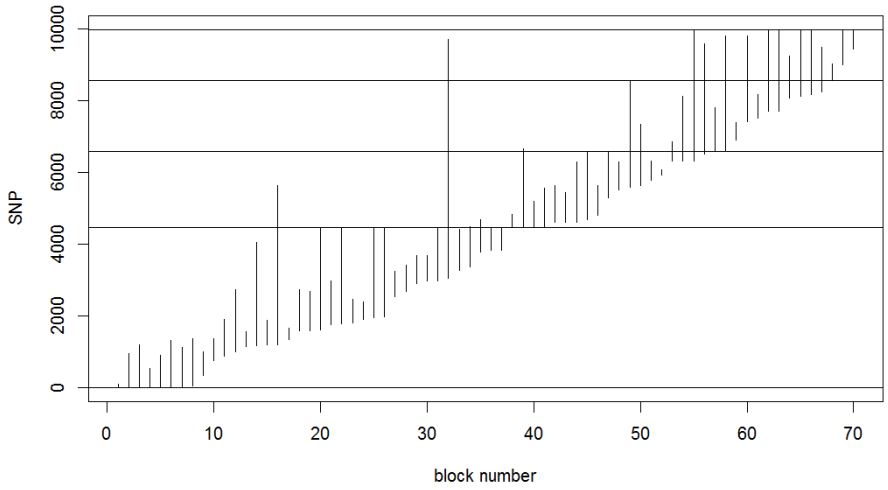


Figure 11: Exemplary output of blocklist_plot for the ex_maze blocklist (cutoff2=3)

9.10 blocklist_plot_xsize()

Parameters: blocklist, cutoff2=5, bound_weighted=TRUE, type="snp"

Plot of the blocks with width according to the number of haplotypes in the respective block. Location according to the y-axis. Additionally recombination hotspots are indicated by horizontal lines. **Cutoff2** is the minimum number of blocks to end to mark a position as a hotspot and **bound_weighted** scales blocks according to the size of the block.

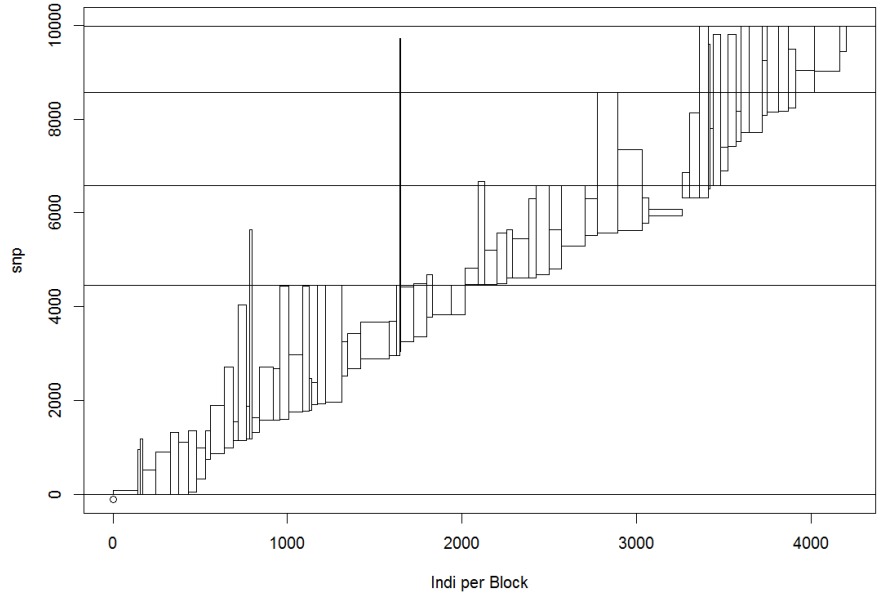


Figure 12: Exemplary output of blocklist_plot_xsize for the ex_maze blocklist (cutoff2=3)

10 Acknowledgements

This package was developed in the context of the BMBF project “MAZE – Accessing the genomic and functional diversity of maize to improve quantitative traits” (Grant ID 031B0195).

Additional thanks goes to the Research Training Group 1644 “Scaling Problems in Statistics” for financing travelling and Manfred Mayer for proofreading of this manuscript.

B Guidelines to MoBPS

MOBPS

Modular Breeding Program Simulator

Torsten Pook, Martin Schlather, Henner Simianer

OCTOBER 29, 2019

UNIVERSITY OF GOETTINGEN
Animal Breeding and Genetics Group

Table of Contents

1 General 4

2 Installation 4

3 Citation 4

4 Creation of the starting population (creating.diploid()) 5

4.1 Importing/Generating of a genetic dataset 5

4.2 Importing a genetic map 6

4.3 Simulating/Generating the genetic architecture underlying each trait 6

4.3.1 Custom-made genetic architectures 7

4.3.2 Predefined genetic architectures 8

4.3.3 Correlated Traits 8

4.4 Position of Markers 8

5 Simulation of breeding processes (breeding.diploid()) 9

5.1 General setup 9

5.2 Control of heritability, breeding values, genotypes and phenotypes 10

5.3 Breeding value estimation 11

5.3.1 Direct approach with known heritability 12

5.3.2 Bayesian approaches (BGLR) 12

5.3.3 GBLUP (EMMREML) 13

5.3.4 GBLUP (sommer) 13

5.3.5 GBLUP (rrBLUP) 13

5.3.6 Pedigree-based (breedR) 13

5.3.7 Parent/Grandparent mean 13

5.3.8 Own function 13

5.3.9 Calculating marker effects & GWAS 14

5.3.10 Calculation of reliabilities 14

5.4 Selection techniques & mating strategies 14

5.4.1 Multiple traits 15

5.4.2 Higher procreation of genetically favored individuals 15

5.4.3 Maximum number of offspring per individual 16

5.4.4 Plant breeding (no-sexes & selfing & DH-production & cloning) 16

5.4.5 Generate offspring for all sire combination 16

5.4.6 Targeted/Fixed mating/Manual selection of individuals 16

5.4.7 Gene-Editing 16

5.4.8 Optimum genetic contribution 17

5.5 Genetic architecture 17

5.6 Other 17

5.6.1 Culling / Death 17

5.6.2 Parameter for target-mating with J.W.R. Martini 18

5.6.3 Allele-frequency per generation 18

5.6.4 Set a random seed 18

5.6.5 save.recombination.history 18

5.7 Storage & computing time 18

5.7.1 Reducing the size of the population list 19
5.7.2 Reducing memory needs in the BVE 19
5.7.3 Inverting G using miraculix 19
5.7.4 On-the-fly calculation of haplotypes 19

6 Exporting information from the population-list (get.XXXX) 20

6.1 get.genos 20

6.2 get.haplos 20

6.3 get.bv / get.bve / get.pheno / get.reliability / get.selectionindex 21

6.4 get.recombi 21

6.5 get.pedigree (1/2/3) 21

6.6 get.cohorts 22

6.7 get.class 22

6.8 get.genotyped 23

6.9 get.time.point 23

6.10 get.creating.type 23

6.11 get.cullingtime 24

6.12 get.individual.loc 24

6.13 get.vcf 24

6.14 get.pedmap 24

6.15 get.database 25

7 Importing information to the population-list 25

7.1 Insert.bve 25

8 Data structure of the population list 26

8.1 $info 26

8.2 $breeding 28

8.2.1 Storage per generation 28
8.2.2 Storage per individual 28

9 Utility functions 29

9.1 bv.development 29

9.2 bv.development.box 30

9.3 Kinship.development 31

9.4 Kinship.emp / kinship.emp.fast 31

9.5 Kinship.exp 32

9.6 analyze.population 32
9.7 new.base.generation 33
9.8 creating.trait 33
9.9 ensembl.map 33
9.10 compute.costs 34
9.11 compute.costs.cohorts 34
9.12 summary 35
9.13 pedmap.to.phasedbeaglevcf 35
10 Memory and computation times 36
11 List of input parameters in breeding.diploid() 36
12 List of input parameters in creating.diploid() 43
13 List of datasets included in the package 46
14 User-interface 47
15 Commonly used word definitions 54
16 Exemplary scripts 54
16.1 Simulation of a MAGIC population in maize 54
16.2 Simulation of Introgression on blue eggshell QTL 55
16.3 Simulation of gene editing in a cow breeding program 58
16.4 Simulation of a base population with a hard sweep 59
References 60
17 Acknowledgements 62

1 General

MoBPS is an R-package to simulate complex and large scale breeding programs with focus on livestock and crop populations. Simulations are performed on an individual basis. MoBPS is a versatile tool, providing standard procedures applied in animal and plant breeding like GBLUP and OGC, but also allowing to use own selection schemes while still controlling the simulation of phenotypes, meiosis and costs of the simulated scheme. The actual process of the simulation can be split up into two steps: the creation of a starting population and the simulation of breeding processes.

As it is our goal to provide a lot of flexibility while performing the simulation, there is a need of many parameters – luckily only a few of those will be needed for most simulations. For a better understanding of the workflow required to set up a simulation, we refer to section 16 for exemplary simulations. For a list of all input parameters and possible initializations, we refer to section 11.

For questions regarding the tool or how to set up your simulation feel free to contact me (Torsten.pook@uni-goettingen.de). We are always happy for questions as it really helps improve the tool and its documentation. For quick reply, it would help to provide a small example of our problem.

In addition, we are currently in the process of developing a graphical interface for the R-package (available at www.mobps.de). Note that this interface is still in active development and not part of the MoBPS paper. The use of the interface is still encouraged, but for major projects we highly recommend (and are looking for) close collaboration for the further development.

2 Installation

The current version of MoBPS requires R 3.0 or higher. We highly recommend the use of the packages RandomFieldsUtils (version 0.5.9+) and miraculix (0.9.7+) as they significantly reduce computing time when working with a high number of markers and individuals. All functionality is available without both package, but simulations can be significantly slower. For direct install via GitHub use the following line of code (this requires the R-package devtools):

```
devtools::install_github("tpook92/MoBPS", subdir="pkg")
```

This option is currently not available for miraculix and RandomFieldsUtils. RandomFieldsUtils is available on CRAN, miraculix will hopefully soon follow. As updates to CRAN are only possible every few month we highly recommend to use the version available on GitHub itself. Packages can be downloaded at https://github.com/tpook92/MoBPS and directly install via the R function *install.packages()*. Usage was tested on Linux and Windows (under windows set type="source", repo=NULL). The usage on Mac OS is currently not recommended. Commonly used Ensembl-maps (Zerbino et al. 2017) are available in the associated R-package MoBPSmaps.

```
devtools::install_github("tpook92/MoBPS", subdir="pkg-maps")
```

For Windows the installation of Rtools is required. RandomFieldsUtils does require the package spam.

3 Citation

There is currently no paper published on our R-package. This will hopefully soon change. For so long we suggest using following to citations for the R-packages MoBPS and miraculix:

```
@Manual{,
```

```
    title = {MoBPS: Modular Breeding Program Simulator},
    author = {Torsten Pook},
    year = {2019},
    note = { Available at https://github.com/tpook92/MoBPS; R-package versi
on 1.4.15},
  }

  @Manual{,
    title = {miraculix: Statistical Functions for Animal Breeding},
    author = {Malena Erbe and Martin Schlather and Florian Skene and Alexan
der Freudenberg},
    year = {2019},
    note = { Available at https://github.com/tpook92/MoBPS; R-package versi
on 0.9.6},
  }
```

4 Creation of the starting population (*creating.diploid()*)

The input for the simulation of a breeding process is a population list. This list is created via *creating.diploid()*.

We provide exemplary genetic maps for some common species, which can be selected via the parameter **template.chip**. Note that primary the number of chromosomes and their genetic length is imported at this step (especially not real markers with known allele frequencies, effects or base pairs). The maps provided via **template.chip** are “cattle” (Ma et al. 2015), “pig” (Rohrer et al. 1994), “chicken” (Groenen et al. 2009), “sheep” (Prieur et al. 2017) and “maize” (Lee et al. 2002). Alternatively, a genomic map can be inserted via the parameter **map** with exemplary maps being provide in the package itself or via import from Ensembl (section 9.9).

4.1 Importing/Generating of a genetic dataset

In case one has haplotype data for the founders/starting population this can be imported via the parameter **dataset** in form of a haplotype dataset:

```
> dataset
      Indi1Haplo1 Indi1Haplo2 Indi2Haplo1 Indi2Haplo2 Indi3Haplo1 Indi3Haplo2 Indi4Haplo1 Indi4Haplo2
SNP1            1           1           1           1           0           1           1           0
SNP2            0           1           1           0           1           0           1           1
SNP3            0           0           0           1           1           0           0           1
SNP4            1           1           1           1           0           0           1           1
SNP5            0           1           0           1           0           1           1           1
SNP6            0           1           0           0           0           1           1           1
SNP7            1           0           1           1           1           1           1           0
SNP8            0           0           0           1           0           1           0           1
SNP9            1           1           0           0           0           0           0           1
SNP10           0           1           0           0           1           1           1           0
```

Datasets can also be imported via entering the path of the vcf file in the parameter **vcfpath**. The R-package vcfR is needed for this. Otherwise, a dataset can be generated by setting the number of SNPs (**nsnp**) and individuals (**nindi**) – we offer four possible modes to simulate starting haplotypes ("all0", "allhetero", "random", "homorandom") leading to haplotypes (000…/000…, 000…/111…,X_1 X_2 X_3… /X_4 X_5 X_6 … with X_i~B(**freq**) X_1 X_2 X_3 … /X_1 X_2 X_3… with X_i~B(**freq**)). On default "random" is used. The allele frequencies for each marker are enter via the parameter **freq** and are sampled from a beta distribution with **beta_shape1** and **beta_shape2** (default: 1,1; leading to a uniform distribution of allele frequencies.

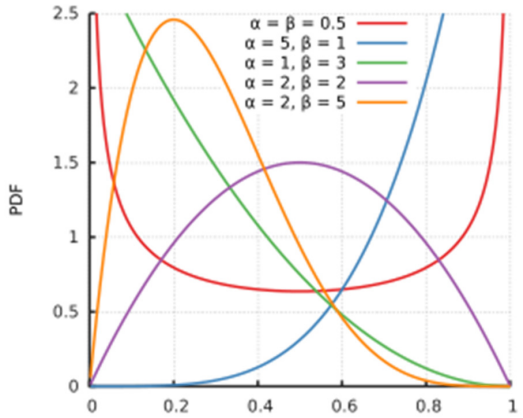


Figure 1: https://en.wikipedia.org/wiki/Beta_distribution

To generate an LD and haplotype structure without using real data we recommend to start with one of the simple datasets and simulate some random/non-random mating generations using *breeding.diploid()* (cf. section 5). Wrapper functions for the automatic generation of those base populations in MoBPS are planned but not yet implemented.

If a vcf file is used for data import or a map is provided, the chromosome, marker name and base pair position are automatically imported. Alternatively, those can be provided via **chr.nr**, **bp** and **snp.name**. By doing this, multiple chromosomes can be inputted jointly. If no input for **chr.nr** is provided all markers are assumed to be on the same chromosome (for more on the usage of a genetic map, we refer to section 4.2).

In case more markers are to be added to and existing population list set **add.chromosome** to TRUE and repeat the previous process with the population list as an additional input.

To specify the sex of each sample either assign each individual a probability to be female (**sex.quota**) or alternatively use a vector (**sex.s**) assigning each individual its sex (M=1, F=2).

4.2 Importing a genetic map

The user can provide a genetic map of up to five colums via the parameter **map**. The first column contains the chromosome of the respective marker, the second column the name of the marker, the third column the physical position of the marker, the forth column the position in centimorgan and the last column the allele frequency in the population. All values not provided are automatically set to NA and values are used as input for **chr.nr**, **snp.name**, **bp**, **snp.position**, **freq**.

Alternatively maps can be imported via *ensembl.map()*. For more on this we refer to Chapter 9.9.

4.3 Simulating/Generating the genetic architecture underlying each trait

As the manual input of effects can be tiring, we provide some automated procedures to simulate some common effect structures (additive, dominant, qualitative and quantitative epistasis) – if you do not need more, you can just skip to section 4.3.2).

Note that this is the generation of an actual genetic value that is underlying each individual of the population. In reality you cannot observe this, as traits will be caused by far more complex interactions

and effects are not known. This, on the other hand, enables opportunities to evaluate a model fit given a known structure (e.g. GWAS hits can be compared to actual effect markers instead of previously identified markers or similar). Traits can be named via the parameter **trait.name**.

4.3.1 Custom-made genetic architectures

To simulate a custom-made genetic architecture we allow for effects caused by one (**real.bv.add**), two (**real.bv.mult**) or more SNPs (**real.bv.dice**). These effects can either be added directly while using *creating.diploid()* or added later using *creating.trait()*. To delete previously existing effects set **replace.real.bv** to TRUE. For multiple traits use lists as inputs for all parameters in this section with each list element containing information for one trait.

Input structure for the first two is a matrix with each row coding a single effect:

```
> real.bv.add
      SNP chromosome Effect 0 effect 1 effect 2
[1,] 120          1      -1.0       0.0         1
[2,]  42          5       0.0       0.0         2
[3,]  17         22       0.1       0.1         0
```

real.bv.add should be able to model any additive or dominance effects of single markers.

```
> real.bv.mult
     First SNP First chromosome Second SNP Second chromosome effect 00 effect 01 effect 02 effect 10 effect 11 effect 12 effect 20 effect 21 effect 22
[1,]       144               1        145                 1      1.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
[2,]         6               3        188                 5      0.37      0.16      1.33      1.49      1.58      2.51      0.38      2.12      0.98
[3,]         5              17          1                10      1.18      2.60      0.18      1.74      0.69      1.39     -1.21      0.96      1.94
```

real.bv.mult should be able to model any epistatic interaction between two markers.

To simulate even more complex effect structures use the parameter **real.bv.dice** allowing the modelling of effects caused by more than two SNPs.

Input for **real.bv.dice** is a list containing a list of all locations and a list of all effects:

```
> real.bv.dice
$location
$location[[1]]
     SNP chromosome
[1,]  11          1
[2,]  12          1
[3,]  16          4

$location[[2]]
     SNP chromosome
[1,]  14          2
[2,]  77          6
[3,]  15          9


$effects
$effects[[1]]
 [1]  1.8212212  1.5939013  1.9189774  1.7821363  1.0745650 -0.9893517  1.6198257  0.9438713  0.8442045 -0.4707524  0.5218499  1.4179416  2.3586796
[14]  0.8972123  1.3876716  0.9461950 -0.3770596  0.5850054  0.6057100  0.9406866  2.1000254  1.7631757  0.8354764  0.7466383  1.6969634  1.5566632
[27]  0.3112443

$effects[[2]]
 [1]  0.29250484  1.36458196  1.76853292  0.88765379  1.88110773  1.39810588  0.38797361  1.34111969 -0.12936310  2.43302370  2.98039990  0.63277852
[13] -0.04413463  1.56971963  0.86494540  3.40161776  0.96076000  1.68973936  1.02800216  0.25672679  1.18879230 -0.80495863  2.46555486  1.15325334
[25]  3.17261167  1.47550953  0.29005357
```

Each network of interacting markers is giving in the first list (location) and their effects are given in a second list (effects). Effects are sorted in following order: 0…0, 0…01, … 2…2 – resulting in a vector with 3^n elements, where n is the number of markers involved in the effect.

Marker effects assign to positions that currently do not exist (e.g. SNP 100 on chromosome 1 in case chromosome 1 only contains 50 SNPs) are automatically removed from the stored effects unless **remove.invalid.qtl** is set to FALSE.

4.3.2 Predefined genetic architectures

In case of a predefined genetic architecture, all markers are assigned with the same probability to be drawn as effect markers. To exclude markers use a parameter **exclude.snps** containing a vector of all numeric positions of excluded markers. Numbering is consecutively starting with chromosome 1. Note that only markers, that are already included in the dataset, can be chosen as effect markers – so in case of more than one chromosome with no generation via **chr.nr** the effects should be added using *creating.trait()* (cf. section 9.2) or in the last run of *creating.diploid()*.

The number of additive (**n.additive**) and dominate (**n.dominant**) QTL as well as effects caused by qualitative (**n.qualitative**) and quantitative (**n.quantitative**) epistasis can be included directly. To assign the variance, one can input a vector containing the variance for each effect (**var.additive, var.dominant,…).** On default, each variance is set to 1 and effects are drawn from a Gaussian distribution.

Qualitative epistasis is simulated by drawing 3 random effects for both involved markers, taking the absolute values from those, sorting them from low (0) to high (2) and multiplying those effects with each other. By this, we obtain the lowest effect for 00 and the highest for 22 with selection for alternative allele to be beneficial in all cases.

Quantitative epistasis is simulated by drawing 9 random effects and assigning the absolute values of two of those to the corner 02 and 20. All other combinations are assigned the minus absolute values of drawn random number.

To simulate more than one trait use vectors for **n.**XXX and lists for **var.**XXX instead.

4.3.3 Correlated Traits

To generate correlated quantitative traits selected the traits that should be correlated via **shuffle.traits** and provide the needed correlation in **shuffle.cor**. Note that QTLs are then assigned to the same markers to get correlations independent of the underlying LD structure. To set a correlation for traits with no underlying QTL, use **new.breeding.correlation**. As simulation is done via sampling from a Gaussian distribution and genetic traits do not fulfil all requirements of a dependent multivariate Gaussian distribution (which is here used to model dependency), the obtained resulting correlations can be different to the correlation set in **new.breeding.correlation** if non-QTL traits have correlations with QTL-traits. We are currently working on alternatives for this.

4.4 Position of Markers

For our simulations, the physical position in base pairs does not really matter as we are interested in a position in Morgan internally. We assume points of recombination to be distributed according to a Poisson distribution. On default, we assume markers to be equidistant with the total length of the chromosome in Morgan given by the parameter **chromosome.length** (default: 5). When performing a joint generation of multiple chromosomes with different sizes enter a vector instead. If non of the following options for the position of each marker are provides, markers are assumed to be equidistant. This will also minimize computation time and therefore should only be changed if needed.

Based on the physical positions entered in the parameter **bp** the position in Morgan can be derived by providing a conversion rate in **bpcm.conversion**. Note that tools like BEAGLE (Browning et al. 2018)

assume 100.000.000 bp per Morgan. For chicken, we would recommend the use of 30.000.000 bp per Morgan.

Another way of entering the genetic position (in M) is via the parameter **snp.position** manually. Scaling can be performed internally by activating the parameter **position.scaling** and the **chromosome.length**. In addition, one should input a value for the number of base pairs before and after the last position (**length.before**, **length.behind**). Both should be chosen to be larger zero (default: 5) if scaling is performed.

For some applications, the recombination rate might not be the same for all individuals (e.g. male/female differences). To input an additional recombination map enter alternative positions in the parameter **add.architecture**. You can select which architecture is used for every parent in the actual simulation process.

5 Simulation of breeding processes (*breeding.diploid()*)

To perform the actual simulation of matings the function *breeding.diploid()* is used. Especially for that step, the sheer amount of different options can be deterring – in reality only a few parameters will actually be relevant. In this section, we will first discuss absolutely necessary parameters, their default options and afterwards discuss possible deviations. For exemplary simulation we refer to section 16. Note that you most likely can skip through some of the sections if you are not interested in changes in that dimension. There will be a lot of cases where there is the same parameter for the male and female part of the breeding program. We will limit ourselves here to the male parameter (parametername**.m**) – usage of the female version (parametername**.f**) will always be the same. The default setting for the female parameter is to be the same as the male parameter (NOT the default of the male parameter). Exception to this is course the case in when groups of individuals are selected to be used as parents and similar (e.g. **selection.f.cohorts**).

5.1 General setup

The output of *breeding.diploid()* is an updated population list. All newly generated individuals are added as an additional generation – to add them to a previously existing generation set **add.gen** to the generation you want to add to.

The number of newly generated individuals can be chosen via **breeding.size**. Input for this is a numeric value and sex of each offspring is randomly determined via **breeding.sex** (probability of a male offspring). To remove randomness set **breeding.sex.random** to FALSE or input a vector containing the number of new male/female individuals in **breeding.sex** instead.

To control which individuals are used in the mating procedure use the parameter **selection.size** (vector of size 2, containing the number of used male/female individuals). By default only the individuals of the last generation and class 0 (this is usually all and you will realize when this is not the case) are used. Classes can be used to model migration in store groups of different genetic origin but same sex and generation or to just remove individuals from the pool of individuals considered for selection.

To control which individuals are used every cohort generated can be named via **name.cohort** in *breeding.diploid()* & *creating.diploid()*. The individuals used in the selection procedure can be chosen via **selection.m.gen**, **selection.m.database** and **selection.m.cohorts** (paternal side) and same syntax for the maternal side. In the old version of the code, this is equivalent to the use **of best1.from.cohorts**

and **best1.from.groups** that are still alternative input parameters for **selection.m.database** and **selection.m.cohorts**.

To combine individuals to a new cohorts of individuals set **combine** to TRUE. This will generate a new cohort of all selected individuals - do not combine male and females individuals!

On default, the selection of individuals is done at random (For more on this we refer to 3.4).

To select the individuals of which class to consider in the selection procedure, use the parameter **class.m**, containing a vector of all usable classes. To control the class of the new individuals use the parameter **new.class** (default 0). Classes of all cohorts added are automatically added to the vector of considered classes (**class.m**) unless **add.class.cohorts** is set to FALSE.

To generate multiple offspring from the same dam/sire pair set **repeat.mating** to the desired number.

Both the time of the generation of new individuals (**time.point**) and the type of the mating (**creating.type**) can be stored. Both parameters are mostly used internally in the web-based application and are automatically tracked internally.

5.2 Control of heritability, breeding values, genotypes and phenotypes

For each individual, an underlying true genetic value is calculated for each trait. Based on this, phenotypes can be generated. For which individuals to generate new phenotypes can be controlled via **new.bv.observation.gen**, **new.bv.observation.database** and **new.bv.observation.cohorts**. For a quick input of all individuals previously not phenotyped set **new.bv.observation** to “all”, “non_obs”, “non_obs_m” or “non_obs_f” for all or all previously not phenotyped individuals (potential of only one sex). On default, for each individual at most one phenotype is generate. Set **multiple.observation** to TRUE to allow for more than one observation per individual. To generate multiple observations in a single run of *breeding.diploid()* set **n.observation** to that number. Note that the number of times an observation for an individual is generated does matter since the environmental variance will be reduced with each observation as observations are assumed to be independet. To model a correlation between the environmental variances for different traits set provide the desired correlation matrix via **new.phenotyp.correlation** (this can also be done in *creating.diploid()*). For the simulation of correlated genetic values we refer to section 4.3.3.

The environmental variance σ_e can be controlled by the usage of **sigma.e**. This can either be set to a fixed numeric value or be estimated to fulfill a target heritability. For the second possibility, the genetic variance is calculated based on the individuals specified in **sigma.e.gen**, **sigma.e.database** and **sigma.e.cohorts** and the needed environmental variance is then calculated by the usage of a predefined **heritability**. You can also use the environmental variance of the previous simulation by setting **use.last.sigma.e** to TRUE. A manual change of the genetic variance **sigma.g** is not recommended but in principle possible (this will only affect the breeding value estimation). On default it is estimated using all individuals used in the current breeding value estimation (set **forecast.sigma.g** to FALSE to deactivate). To specify which groups to use to estimate σ_g use **sigma.g.gen**, **sigma.g.database**, **sigma.g.cohorts**.

For newly created individuals the phenotype is set to 0. Alternatively, one can it to the mean of the parents or create an observation by setting **new.bv.child** to “mean” or “obs” instead of “zero”. In case of generating individuals via **copy.individual** one can also use “addobs” to import existing observations but also potential generate additional ones via **n.observation**. Estimated breeding values are also kept unless **copy.individual.keep.bve** is set to FALSE.

To select the share of individuals genotyped use the parameter **share.genotyped** or select it manually via **genotyped.s** (in concordance to **sex.s** in *creating.diploid()*). In case an individual is generated via **copy.individual** the genotyping state is keep and the share of previously not genotyped individuals that is now genotyped can be controlled via **added.genotyped**.

In some applications, the genetic value of the individual itself is not of importance – instead the performance of its offspring is of relevancy. To select for which individuals to import offspring phenotypes use **offspring.bve.parent.gen**, **offspring.bve.parent.database** and **offspring.bve.parent.cohorts**. Unless specified in **offspring.bve.offspring.gen**, **offspring.bve.offspring.database** and **offspring.bve.offspring.cohorts** all offspring are considered here.

For better comparison of and between breeding values it is possible to standardizing breeding values before the first generation by activating **standardize.bv**. By this the average breeding value is set to **standardize.bv.level** (default: 100) – for the calculation of this, the average of the individuals in generation **standardize.bv.gen** (default: 1) is used.

Scaling in case of index selection with multiple traits is performed in the selection process itself.

5.3 Breeding value estimation

To perform selection one can perform breeding value estimation. To activate this set **bve** to TRUE. In the simplest case, one has to input which groups to use in the breeding value estimation via the parameters **bve.gen**, **bve.database** and **bve.cohorts**.

As there are a lot of different ways to perform breeding value estimation, we implemented multiple variants:

1. GBLUP with assumed known heritability and direct solving of the mixed model without REML variance component estimation
2. Bayesian approaches implemented in BGLR
3. GBLUP using EMMREML
4. GBLUP using sommer
5. Pedigree-based BLUP via breedR
6. Parent/Grandparent mean
7. Own function

In any case estimated breeding values are entered for all individuals unless **bve.insert.gen**, **bve.insert.database** or **bve.insert.cohorts** directly classifies for which groups breeding values are to be entered. The accuracy of the breeding value estimation is automatically reported unless **report.accuracy** is set to FALSE.

For the calculation of G we offer multiple methods with **computation.A**=”vanRaden” being the default (VanRaden 2008). Alternatives include “kinship”, “CM”, “CE” (Martini et al. 2017) and the non-Z-standardized version of the vanRaden method (“non_stand”). In case “kinship” is selected the depth of the pedigree has to be provided via the parameter **depth.pedigree** (default: 7). Note that these individuals are just used to calculate the A matrix. Individuals used in the actual breeding value estimation still need to be selected via **bve.gen**, **bve.database** and **bve.cohorts**. Individuals with no observed phenotype start with a value of 0. Internally all phenotypes that are exactly 0 are handled as an NA – suppress this by setting **bve.0isNA** to FALSE (note that only methods 1./2./4. are able to handle NAs in the data). To active the use of the single step genomic relationship matrix set **singlestep.active**

to TRUE – otherwise non-genotyped individuals are not considered in the breeding value estimation unless **remove.non.genotyped** is set to FALSE (Legarra et al. 2014).

To not perform statistical breeding value estimation but instead using the phenotypes as breeding value estimates set **phenotype.bv** to TRUE.

As the presence of true effect markers in the dataset might be a strong assumption one can set **remove.effect.position** to not use markers associated with any traits in the breeding value estimation.

To only included individuals with a certain class set **bve.class** to a vector containing all classes to consider.

In case individuals were generated via **copy.individual** (this is especially relevant for the web-based application) each individuals is only used at most once. To consider the same individual multiple times set **bve.avoid.duplicates** to FALSE. Note that cloning will not lead to the same ID.

5.3.1 Direct approach with known heritability

Main advantage of a direct estimation is a massive improvement in computation time as the usually necessary REML estimation takes most of the time. In practice, it might not be realistic but since genetic values are known it is possible. Note that this is still an empirical measure that can change when using different individuals in the estimation process. Especially for bigger populations heritability estimation should not be problematic and is not performed in each breeding value estimation in practice as well. To instead estimate the additive genetic variance using a parental model activate **estimate.add.gen.var**.

In case of missing phenotypes, estimates will be based on (VanRaden 2008) method 2. This will also be used in case of the use of single step. Alternatively one can use rrBLUP based estimates by setting **bve.direct.est** to FALSE. Note that this second version is slower and requires the presence of individuals that are phenotyped and genotyped.

5.3.2 Bayesian approaches (BGLR)

For performing Bayesian methods we are using the R-package BGLR. To activate the usage of BGLR set **BGLR.bve** to TRUE. On default a Reproducing-kernel-hilbert-space is used – alternatively one can use BayesA, BayesB, BayesC by setting **BGLR.model** to “BayesA”, “BayesB”, “BayesC”, “BRR”, or “BL” instead of “RKHS”. Other parameter values will all be chosen according to the defaults of the BGLR package. If you want to test alternative parameter settings or use other methods implemented in BGLR either do the estimation manually (Chapter 5.3.8) or contact the me to add it to the package.

To control the number of the burn-in and iterations use **BGLR.burnin** and **BGLR.iteration**. To deactivate printing of results of interim steps set **BGLR.print** to FALSE (equal to verbose=FALSE in BGLR). On default BGLR will generate some internal files in its computations. To select a path of where to store them chose it via **BGLR.save**. Especially when parallelizing thousands of simulations BGLR tents to crash when the same path is used multiple times. Activating **BGLR.save.random** will hinder this.

5.3.3 GBLUP (EMMREML)

Traditional GBLUP including variance component estimation using REML is performed by using the package EMMREML. To activate the usage set **emmreml.bve** to TRUE. EMMREML does not support missing phenotypes and therefore can only be used if phenotypes for all individuals in the BVE are available (if not use the direct approach, sommer or rrBLUP).

5.3.4 GBLUP (sommer)

Traditional GBLUP including variance component estimation using REML is performed by using the package sommer. To activate the usage set **sommer.bve** to TRUE. Sommer does support missing phenotypes.

To activate the use of the multi-trait model implemented in sommer use **sommer.multi.bve** to TRUE. Note that this will take substantially longer than single trait models.

5.3.5 GBLUP (rrBLUP)

Traditional GBLUP including variance component estimation using REML is performed by using the package rrBLUP. To activate the usage set **rrBLUP.bve** to TRUE. rrBLUP is about 2.5 times as fast as sommer for breeding value estimation.

5.3.6 Pedigree-based (breedR)

Traditional breeding value estimation using pedigree data. To activate set **breedR.bve** to TRUE – especially for bigger populations this is much faster computation wise. Overall heritabilities in breedR seem to be underestimated and accuracies slightly lower. Alternatively, the pedigree-based relationship matrix can also be used in all other methods by setting **computation.A** to “kinship”. This procedure takes about the same time as with other relationship matrices (No usage of high number of zeros in the relationship matrix or the direct inverse of the pedigree matrix).

5.3.7 Parent/Grandparent mean

To use the mean performance of the parents / grandparents as the breeding value use **bve.parent.mean** / **bve.grandparent.mean**. On default breeding value estimates for the parents are used and if those are not available phenotypes. Alternatively on can select to use breeding values only (“bve”), phenotypes only (“pheno”) or genomic values (“bv”) via the parameter **bve.mean.between**.

5.3.8 Own function

Instead of performing breeding value estimation inside of *breeding.diploid()* one can implement his own methodology by exporting all information needed to those computations and inserting own breeding values estimates via the function *insert.bve().*

According code could look like this:

```
# Simulate Phenotypes for generation 4 with heritability 0.4
population <- breeding.diploid(population, heritability = 0.4,
```

```
                                sigma.e.gen = 4,
                                new.bv.observation.gen = 4)
# Export genotypes and phenotypes for generation 4
genos <- get.geno(population, gen = 4)
phenos <- get.pheno(population, gen = 4)

# Here you perform your own method to assign breeding values to each individual
bve <- runif(ncol(genos)) # This is probably not the best technique for this =)

# Import breeding values estimated for generation 4
bves <- cbind(colnames(genos), bve)
population <- insert.bve(population, bves=bves)
```

For details on exporting functions, we refer to section 6. For details on importing function, we refer to section 7.

5.3.9 Calculating marker effects & GWAS

For some applications (e.g. gene editing) it is necessary to identify causal markers. Although marker effects are known in a simulation, in practice one has to identify them. Options here are either a direct calculation of the effect size of each marker based on the computations performed in 5.3.1 (rrBLUP) or the performance of a GWAS-study without correction for population structure. Methods can be activated by setting **estimate.u** or **gwas.u** to TRUE.

In case of a GWAS study one can additionally select the groups used in the study by setting **gwas.gen**, **gwas.database** and **gwas.cohorts** (default is same as for breeding value estimation). As a value for y, one can use the phenotype ("pheno"), true breeding value ("bv") or the estimated breeding value ("bve"). Additionally it might be necessary to standardize the y value by the mean of the group by activating **gwas.group.standard**. Note that this is a basic implementation of GWAS with no correction for population structure or similar.

5.3.10 Calculation of reliabilities

Reliabilities are not derived in any of the used R-packages. In the direct approach (Chapter 5.3.1), they can be derived by setting **calculate.reliability** to TRUE according to (VanRaden 2008).

5.4 Selection techniques & mating strategies

Selection of the individuals for matings in the following generations is of key importance for any breeding program. Especially here, one is limited to the techniques that work in the species one wants to simulate. On default settings, the selection of the new founders is done at random. To use estimated breeding values as a selection criteria set **selection.m** to "function". To ignore the best selected individuals set **ignore.best** to that value – note that this value will be internally subtracted from **selection.size**. E.g. to simulate mating between the top 100 female individuals with the third and fourth best male individual set **selection.size** = c(4,100) and **ignore.best**=c(2,0). To exclude specific individuals from the set of individuals to select from use **reduced.selection.panel.m**. The vector should contain all individuals to use (e.g. 1:10 when selecting from the first ten individuals).

To store details on which individuals were selected, which mating were performed and the currently estimated breeding values activate **store.breeding.totals**.

Selection can be performed based on the phenotype, genetic value or the breeding value estimates. To select what to use set **selection.criteria.type** to “pheno”, “bv” or “bve” (default: “bve”).

5.4.1 Multiple traits

When working with multiple traits, the selection of the best individuals is typically done by the use of a combination of those traits. All single values can either be added up directly (**multiple.bve**=”add”) or one can use a selection index just accounting for the ranking (**multiple.bve**=”ranking”). To reduce scaling problems for different traits one can use **multiple.bve.scale.m** to standardize the variance in each trait. Note that this scaling in the cohort mode is for all individuals together whereas in the old selection modes it is done per group (! – needs to be the same!!!). Additionally, each trait can be assigned a weighting via **multiple.bve.weights.m**.

To derive the ideal index based on phenotypic/genotypic variance, reliabilities and economic gains per unit according to (Miesenberger 1997) set **selection.m.miesenberger** to TRUE. Economic gains can be provided in **multiple.bve.weights.m**. On default, the gain has to be provided per standard deviation of the breeding value estimations. Alternatives can be provided in **selection.miesenberger.w** (default: “bve_sd”) and are per unit (“unit”) and per phenotypic standard deviation (“pheno_sd”).

In case reliabilities are not derived (Chapter 5.3.10), they need to be estimated. On default, this is done by dividing the standard deviation of the breeding value estimation by the standard deviation of the phenotypes. Alternatives can be entered in **selection.miesenberger.reliability.est**, as the direct use of the heritability (“heritability”) or the actual calculation according to the correlation between breeding value estimates and true underlying genomic values (“derived”) which is of course not possible in practice but should be the most accurate.

5.4.2 Higher procreation of genetically favored individuals

Genetically favored individuals tend to procreate more often. To model this set a ratio between the likelihood of the best individual to mate compared to the worst individual (in the group of selected individuals) in **best.selection.ratio.m**. This ratio is the ratio between the frequency the best selected individual and the worst selected individual. All other frequencies are then calculated linearly. E.g. in a group of selected individuals with breeding values 105, 103 and 100 with a ratio of 6 the relative frequencies are of 6,4,1 (this just is a linear function – comp for individual 2: (103 – 100)/ (105-100) * (6-1) +1). Criteria behind can be either “bv”, “bve” or “pheno” and can be entered in **best.selection.criteria.m**. To manually enter the probability of each individual in the group of selected individuals input a vector with frequencies for each individual in **best.selection.manual.ratio.m**. Individuals selected are sorted with the individual with the highest estimated breeding value being the first one.

This does not require breeding value estimation and can also be used to simulate slow natural selection processes over thousands of generations.

Higher procreation is also relevant for optimum genetic contribution theory and the use of **ogc** will automatically change these parameters accordingly (section 5.4.8).

5.4.3 Maximum number of offspring per individual

To control the maximum number of times each individual is used for reproduction set **max.offspring**. Either enter a numeric if that boundary is for both sexes or a vector with the first value coding the maximum for male and the second the one for female.

5.4.4 Plant breeding (no-sexes & selfing & DH-production & cloning)

For some applications the sex of an individuals is not relevant (or an individual does not even have a sex). Even though a sex is still stored internal it might be neglectable for the application at hand. In this case, one can allow matings between individuals from the same sex by usage of **same.sex.activ**. The probability to select a female individuals as a parent can be set via **same.sex.sex** (default=0.5). To additionally allow for selfing set **same.sex.selfing** to TRUE. The probability for this mating is the same as any other mating combination.

To perform exclusively selfings, activate **selfing.mating** and selected the probability to use a female parent via **selfing.sex**.

To generate doubled haploid lines active **dh.mating** and selected the probability to use a female parent via **dh.sex**.

To generate and exact copy of an individual to **copy.individual** to TRUE. Instead of simulating the meiosis both chromosome sets of the selected first parent (usually father) will be copied (to copy female individuals use **same.sex.activ** and set the probability to use females to 1 (**same.sex.sex**=1).

5.4.5 Generate offspring for all sire combination

To generate offspring from each possible parental combination set **breeding.all.combination** to TRUE. In case only individuals from one sex are selected sex is ignored when deriving potential matings. **Breeding.size** still has to be set.

5.4.6 Targeted/Fixed mating/Manual selection of individuals

If none of the previously described methods works for your simulation, you can also manually enter a list of all matings that should be performed. For this, use the parameter **fixed.breeding**. To perform targeted mating in the group of the best individuals use **fixed.breeding.best**. Here each row just contains the sex and position in the list of selected individuals. In both cases, an additional column can be added that is coding the likelihood of the offspring being female.

5.4.7 Gene-Editing

With increasing popularity of methods like CRISPR/Cas9 one might be interested in performing gene editing to increase the genetic gain. Gene editing can be activated by setting **gene.editing** to TRUE. The number of edits can be controlled via **nr.edits** and effect markers are picked via usage of the predictions via rrBLUP/GWAS in section 5.3.9. We only count actually performed edits – if an allele is already beneficial, the next best marker is edited instead. Although such a procedure is not possible

in practice, the first integrated way of editing is to edit all selected individuals – this technique is also performed in the approach PAGE (Jenko et al. 2015) and our counter version (Simianer et al. 2018).

As a more realistic scenario, we also allow for the editing of offspring via **gene.editing.offspring**. To only perform editing on male or female individuals set **gene.editing.offspring.sex** / **gene.editing.best.sex** to 1 (male) or 2 (female).

Note that traditionally modelled effects often neglected strongly deleterious mutations and we are here assuming that 100% of all edits to work, all possible offspring will survive the procedure and traits are as simple as designed (usually single marker QTL).

5.4.8 Optimum genetic contribution

To use optimum genetic contribution theory from the group of selected individuals set **ogc** to TRUE. The target increase of the average relationship can be selected via **ogc.cAc**. On default, the increase is minimized. MoBPS is used the traditional formula according to (Meuwissen 1997) and a pedigree based relationship (change via the parameter **computation.A.ogc**). One can also provide on weightings via **best.selection.manual.ratio.m** (Chapter 5.4.2) and/or **fixed.breeding** (Chapter 5.4.6). We are also willing to implemented alternatives if they are needed/wanted.

5.5 Genetic architecture

When simulating meiosis, we are accounting for recombination and mutation. We are assuming that recombination points are Poisson distributed with one expected point of recombination per 1 Morgan. To change this, set **recombination.rate** to the needed value. To not use, a fixed value but a step-function instead use **recom.f.indicator**. Additional genetic architectures can be added the same way as in *creating.diploid()* via **add.architecture**. To select the genetic architecture of recombination for set **gen.archicture.m** to the architecture that should be used for males.

Regarding mutation rates we are assuming that each marker has the same probability for a mutation – this can be changed via **mutation.rate** (default: 10^{-5}). A mutation back to the reference is assigned the probability of **remutation.rate** (default: 10^{-5}). Those values tend to be a lot higher than what you would expect in nature. Depending on the base pair one would expect something around 10^{-8} for a random loci.

Duplications are implemented but the modelling is absolutely adhoc and probably needs refinement – talk to me if you plan to do something in that direction!

5.6 Other

5.6.1 Culling / Death

5.6.1.1 Culling module (Web-interface)

The new culling module is currently only available for cohorts and mainly intended for interface users. Here, parameters settings will take care of itself. To manually execute this in R use **culling.cohort** to select for which cohort to execute the module. The age of the individual can be provided in **culling.time**. The name of the culling reason can be provided in **culling.name**. Additional one can provide two breeding values (**culling.bv1**, **culling.bv2**) and two culling probabilities

(**culling.share1**, **culling.share2**). For all other genomic values the probability of culling this then derived with linear extension.

An index of weighting between traits can be selected via **culling.index** (similar to **multiple.bve.weights.m** – Chapter 5.4.1). On default, no genomic influence is assumed and all individuals are culled with the same probability (**culling.share1**).

5.6.1.2 Old module

Especially for cost calculation it might be necessary to know the time of death for each individual. A group of individuals can be reduced to **reduce.group** with each row coding generation, sex, number of individuals to keep, class. To set the selection criteria use **reduce.group.selection** (default: "random").

5.6.2 Parameter for target-mating with J.W.R. Martini

Don't think anyone needs documentation here – code is specific to the planned paper with Johannes and does not generalize (quick and dirty - https://github.com/Droogans/unmaintainable-code)

martini.selection / Special.comp / Special.comp.add / Max_auswahl / Predict.effects / SNP.density / Use.effect.markers / Use.effect.combination

5.6.3 Allele-frequency per generation

To store the frequency of each allele per generation activate **store.effect.freq**.

5.6.4 Set a random seed

For repeatability it might be helpful to set a random seed in R. This can be done via the parameter **randomSeed** or directly performed in R using *set.seed().*

5.6.5 save.recombination.history

To store the time of occurance of each point of recombination activate **save.recombination.history**. This has to be done starting with the first generation and currently crashes after setting a new founder population (Currently nobody needs it! – but should be an easy fix!)

5.7 Storage & computing time

Computing time for the whole simulation (population$info$comp.times), the steps of the breeding value estimation (population$info$comp.times.bve) and the generation of new individuals (population$info$comp.times.generation) are stored in the population list. If this is not required you can deactive this by setting **store.comp.times**, **store.comp.times.bve** and **store.comp.times.generation** to FALSE. Unless **verbose** is set to FALSE you should automatically receive notifications on the current step of the algorithm and computing times of individual steps. Activation of **Rprof** can provide even more information.

5.7.1 Reducing the size of the population list

Especially when simulating populations with lots of markers, individuals and/or generations, data storage can become a problem. As the internal structure of a population list is complex and manually deleting of things is not recommended – use following settings in *breeding.diploid()* instead:

delete.haplotypes: Vector containing all generations for which haplotypes no longer need to be stored (note that only founder generations are stored anyway – everything else is calculated on-the-fly)

delete.same.origin: Merge two adjacent segments with the same founding haplotypes (deletion of a recombination point with no influence)

delete.individuals: Vector containing all generations for complete deletion– to only delete one sex use **delete.sex** (vector contain sex to delete – 1 (male), 2 (female). Especially when the number of recombinations stored per individual, becomes bigger this is of relevancy.

5.7.2 Reducing memory needs in the BVE

To calculate the genomic relationship matrix, one has to perform matrix multiplication of a matrix containing n*p entries (individuals x markers). Note that for every entry only 2 bits are needed when using miraculix. Nevertheless, this can become extremely big – to reduce this, one can perform the calculation of G sequentially and only load a part of Z into memory at any time.

To activate this, set **sequenceZ** to TRUE. The number of markers in memory can be controlled via **maxZ** (default: 5000). Alternatively, one can put **maxZtotal** to control the total number of entries instead. As this is increasing the computation time, we first recommend to activation of miraculix. As long as miraculix is installed this will happen automatically unless the parameter **miraculix** in *creating.diploid()* is set to FALSE. For more on this we refer to section 9.12.

To speed up commutation one can use multiple cores by the usage of **miraculix.cores** (default: 1) or in case miraculix is not active **ncore**. The backend outside of miraculix is using doParallel and mclapply but doMPI is supported as well – we highly recommend the use of miraculix instead.

Setting **fast.compiler** to TRUE will additionally activate a just-in-time-compiler (enableJIT(3)).

To save computation speed in the GWAS, one can use **approx.residuals** – this does not influence the order of the predicted effect markers but will influence p-values slightly.

5.7.3 Inverting G using miraculix

The inversion of $(G + I_n \cdot \lambda)$ can take a lot of time, is numerically unstable and might not even be possible at all if the matrix is not invertible at all. Instead of the standard cholesky procedure using *chol2inv(chol())*, the inversion can also be done in RandomFieldsUtils/miraculix by activating **chol.miraculix**. Leading to slightly reduced computing times – but also includes screening for semi definite matrices and an automatically changed algorithm, if needed, and thus proceeding without error.

5.7.4 On-the-fly calculation of haplotypes

To save memory, haplotypes are calculated on-the-fly. For this, the location of each recombination point (between which markers) has to be stored. In case one is working with equidistant markers, it

basically takes no time. For other cM-positions it might increase computation speed to provide a function that derives the last marker in front of a certain position in Morgan. This function can be entered via **import.position.calculation**. Only in extreme cases (lots of markers) this should even matter!

6 Exporting information from the population-list (get.XXXX)

Most of the data stored in a population list is highly compressed since saving haplotypes of all individuals of the dataset for all generations would often exceed most local machines or even servers. For some applications (especially if one wants to perform his own fancy simulations without contacting the author and asking him to extent the package) it might be useful to understand the data structure behind. For that, we refer to section 8. In most cases, using our predefined export functions should be enough:

In what individuals one is interested in can be controlled by usage of the parameter **gen**, **database** and/or **cohorts**. Here **cohorts** and **gen** are vectors containing all generations/cohort to include whereas **database** contains a matrix with each row coding a group to export:

```
> database
     Generation sex
[1,]          1   2
[2,]          5   1
```

This **database** will export the information for all female individuals from generation 1 and all male individuals from generation 5. If require a third and fourth column in **database** can be added to indicate the first and last individual of the group to consider. MoBPS will automatically add this to be the full group when *get.database()* is used.

6.1 get.genos

This function will export genotypes. To additionally output the base pair of the minor/major allele set the parameter **export.alleles** to TRUE. Each column contains one genotype with column names indicating sex, individual number and generation.

```
> genos <- get.geno(population,gen=3)
> genos[1:5,1:10]
         M1_3 M2_3 M3_3 M4_3 M5_3 M6_3 M7_3 M8_3 M9_3 M10_3
Chr1SNP1    1    2    1    0    2    1    2    2    2     1
Chr1SNP2    1    0    1    0    2    1    1    2    1     0
Chr1SNP3    1    0    1    1    0    1    0    1    0     1
Chr1SNP4    0    0    1    1    1    1    0    0    0     1
Chr1SNP5    1    0    1    2    1    1    0    0    0     0
```

6.2 get.haplos

This function will export haplotypes. To additionally output the base pair of the minor/major allele set the parameter **export.alleles** to TRUE. Each column contains one haplotype with column names indicating sex, individual number, chromosome set (currently only diploid individuals) and generation.

```
> haplos <- get.haplo(population,gen=3)
> haplos[1:5,1:10]
         M1_3_set1 M1_3_set2 M2_3_set1 M2_3_set2 M3_3_set1 M3_3_set2 M4_3_set1 M4_3_set2 M5_3_set1 M5_3_set2
Chr1SNP1         0         1         1         1         0         1         0         0         1         1
Chr1SNP2         0         1         0         0         0         1         0         0         1         1
Chr1SNP3         1         0         0         0         1         0         0         1         0         0
Chr1SNP4         0         0         0         0         1         0         1         0         1         0
Chr1SNP5         1         0         0         0         1         0         1         1         1         0
```

6.3 get.bv / get.bve / get.pheno / get.reliability / get.selectionindex

These functions will export the true underlying breeding value ("bv"), the estimated breeding value ("bve"), the phenotype ("pheno"), the reliability for each breeding value estimation/trait ("reliability") and the selection index used in the last selection procedure ("selectionindex").

```
> bv <- get.bv(population,gen=1)
> bve <- get.bve(population,gen=1)
> pheno <- get.pheno(population,gen=1)
> bve[1:5]
[1] 45.95833 48.85317 31.53675 35.73686 49.55310
> bv[1:5]
[1] 33.35516 47.52439 31.92523 59.13089 37.96659
> pheno[1:5]
[1] -22.230518  67.241256  -9.337895 -16.056936 101.497754
```

6.4 get.recombi

This function will export all points of recombination and the genetic origin of each segment. The structure here is a list of 4 elements with elements 1 (paternal) and 2 (maternal) containing recombination points and elements 3 (paternal) and 4 (maternal) containing the genetic origin.

Each row is coding the genetic origin between two points (generation, sex, individual number, chromosome set). In the example provided this would mean that the segment between 0.000 and 0.218 of the paternal chromosome originates from the second chromosome set of the 532nd female individual of the first generation.

Note that in addition to all recombination points the start and end points of chromosomes are also exported.

```
> recombi <- get.recombi(population, gen=3)
> recombi[[1]][[1]]
 [1]  0.0000000  0.2183368  0.3517032  0.6187816  1.0718667  1.3089574  1.6053402  1.6390138  1.6627053  1.9827624  2.0846120
[12]  2.1364282  2.4399836  3.0289287  3.7351480  4.0304151  4.6348112  5.5724172  5.9576469  6.5091164  6.9677451  7.4578594
[23]  8.5493007  8.8105940  8.9809813  9.2018383 10.0248883 11.4693043 11.6175350 11.6398517 12.6182508 13.4061942 13.5190658
[34] 13.8967908 14.4158473 14.4829801 14.7182578 15.1508620 15.1545097 15.5787925 15.7266202 15.8199412 16.3273204 17.1619292
[45] 17.3799130 17.7438021 18.1580695 19.0026216 19.8908821 20.1429153 21.3074407 22.2018853 22.5145334 22.8492019 22.8980965
[56] 23.2768180 23.4729906 23.8671758 24.1965660 24.2540496 24.5550846 24.9169786 25.2573038 25.3671158 25.5634302 25.7719966
[67] 25.9989692 26.2255921 26.8175851 26.8552645 27.0569816 27.3228323 27.7300196 27.7441943 28.0331752 28.0909684 28.0925820
[78] 28.4657455 28.7984996 29.0263514 29.1274251 29.2751559 29.3417943 29.5015208 29.6375127 29.8279188 30.0385607 30.1115719
[89] 30.1383062 30.2662008 30.2851334 30.4432174
> recombi[[1]][[3]][1:5,]
     [,1] [,2] [,3] [,4]
[1,]    1    2  352    2
[2,]    1    2  352    1
[3,]    1    1   78    1
[4,]    1    2  352    2
[5,]    1    2  352    1
> recombi[[1]][[5]]
[1] "M1"
```

6.5 get.pedigree (1/2/3)

This function will export the pedigree. Individuals are coded by sex, individual number and generation.

```
> ped <- get.pedigree(population, gen=12)
> ped[1:10,]
      offspring father     mother
 [1,] "M1_12"   "M2214_11" "W1776_11"
 [2,] "M2_12"   "M2052_11" "W1125_11"
 [3,] "M3_12"   "M1529_11" "W1904_11"
 [4,] "M4_12"   "M1712_11" "W1256_11"
 [5,] "M5_12"   "M221_11"  "W326_11"
```

Instead of character string with “M”/”W” indicting sex, one can also directly export a table with 9 colums indicating Sex (1/2), Generation(1,2,3,…) and individidal number (1,2,3,…) in a numeric format by setting **raw** to TRUE.

To export grandparents use *get.pedigree2()*, to get both *get.pedigree3()*. In *get.pedigree2()* one can additionally export the share of the genome inherited by which grandparent by setting the parameter **shares** to TRUE.

```
> ped <- get.pedigree2(population, gen=12)
> ped[1:5,]
     offspring grandfatherf grandmotherf grandfatherm grandmotherm
[1,] "M1_12"   "M734_10"    "W2135_10"   "M2342_10"   "W313_10"
[2,] "M2_12"   "M1188_10"   "W1436_10"   "M489_10"    "W539_10"
[3,] "M3_12"   "M1702_10"   "W1860_10"   "M413_10"    "W1234_10"
[4,] "M4_12"   "M1560_10"   "W1211_10"   "M1649_10"   "W254_10"
[5,] "M5_12"   "M1649_10"   "W2093_10"   "M1593_10"   "W2390_10"
> ped <- get.pedigree3(population, gen=12)
> ped[1:5,]
     offspring father     mother     grandfatherf grandmotherf grandfatherm grandmotherm
[1,] "M1_12"   "M2214_11" "W1776_11" "M734_10"    "W2135_10"   "M2342_10"   "W313_10"
[2,] "M2_12"   "M2052_11" "W1125_11" "M1188_10"   "W1436_10"   "M489_10"    "W539_10"
[3,] "M3_12"   "M1529_11" "W1904_11" "M1702_10"   "W1860_10"   "M413_10"    "W1234_10"
[4,] "M4_12"   "M1712_11" "W1256_11" "M1560_10"   "W1211_10"   "M1649_10"   "W254_10"
[5,] "M5_12"   "M221_11"  "W326_11"  "M1649_10"   "W2093_10"   "M1593_10"   "W2390_10"
```

6.6 get.cohorts

This function extracts all existing cohorts from the population list. Set **extended** to TRUE to also extract further information on the cohorts:

```
> get.cohorts(population)[1:5]
[1] "Founder_M"      "Founder_W"      "Cows"           "Bulls"          "Selected_bulls"
> get.cohorts(population, extended=TRUE)[1:5,]
     name           generation male individuals female individuals class position first male position first female time point creating.type
[1,] "Founder_M"    "1"        "50"            "0"               "0"   "1"                "0"                  "0"        "0"
[2,] "Founder_W"    "1"        "0"             "50"              "0"   "0"                "1"                  "0"        "0"
[3,] "Cows"         "2"        "0"             "50"              "1"   "1"                "1"                  "1"        "2"
[4,] "Bulls"        "2"        "50"            "0"               "1"   "1"                "51"                 "1"        "2"
[5,] "Selected_bulls" "3"      "10"            "0"               "2"   "1"                "1"                  "1"        "1"
```

6.7 get.class

This function extracts the class of each individual:

```
> get.class(population, gen=1:2)
 M1_1  M2_1  M3_1  M4_1  M5_1  M6_1  M7_1  M8_1  M9_1 M10_1 M11_1 M12_1 M13_1
    0     0     0     0     0     0     0     0     0     0     0     0     0
M14_1 M15_1 M16_1 M17_1 M18_1 M19_1 M20_1 M21_1 M22_1 M23_1 M24_1 M25_1 M26_1
    0     0     0     0     0     0     0     0     0     0     0     0     0
```

…

```
W42_1 W43_1 W44_1 W45_1 W46_1 W47_1 W48_1 W49_1 W50_1  M1_2  M2_2  M3_2  M4_2
    0     0     0     0     0     0     0     0     0     1     1     1     1
 M5_2  M6_2  M7_2  M8_2  M9_2 M10_2 M11_2 M12_2 M13_2 M14_2 M15_2 M16_2 M17_2
    1     1     1     1     1     1     1     1     1     1     1     1     1
M18_2 M19_2 M20_2 M21_2 M22_2 M23_2 M24_2 M25_2 M26_2 M27_2 M28_2 M29_2 M30_2
    1     1     1     1     1     1     1     1     1     1     1     1     1
```

6.8 get.genotyped

This function extracts if an individual is genotyped or not (mostly relevant for cost calculation and/or use in Single Step GBLUP (Legarra et al. 2014).

```
> get.genotyped(population, gen=6)
 M1_6  M2_6  M3_6  M4_6  M5_6  M6_6  M7_6  M8_6  M9_6 M10_6 M11_6 M12_6
FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
M13_6 M14_6 M15_6 M16_6 M17_6 M18_6 M19_6 M20_6 M21_6 M22_6 M23_6 M24_6
 TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
M25_6 M26_6 M27_6 M28_6 M29_6 M30_6 M31_6 M32_6 M33_6 M34_6 M35_6 M36_6
FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
M37_6 M38_6 M39_6 M40_6 M41_6 M42_6 M43_6 M44_6 M45_6 M46_6 M47_6 M48_6
 TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
```

6.9 get.time.point

This function extract the time point of generation – this is mostly applicable when using the web-based application since there the first possible time point of generation is automatically calculated.

```
> get.time.point(population , gen=1:2)
 M1_1  M2_1  M3_1  M4_1  M5_1  M6_1  M7_1  M8_1  M9_1 M10_1 M11_1 M12_1 M13_1 M14_1 M15_1
    0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
M16_1 M17_1 M18_1 M19_1 M20_1 M21_1 M22_1 M23_1 M24_1 M25_1 M26_1 M27_1 M28_1 M29_1 M30_1
    0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
```

…

```
W41_1 W42_1 W43_1 W44_1 W45_1 W46_1 W47_1 W48_1 W49_1 W50_1  M1_2  M2_2  M3_2  M4_2  M5_2
    0     0     0     0     0     0     0     0     0     0     1     1     1     1     1
 M6_2  M7_2  M8_2  M9_2 M10_2 M11_2 M12_2 M13_2 M14_2 M15_2 M16_2 M17_2 M18_2 M19_2 M20_2
    1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
```

6.10 get.creating.type

This function extracts the creating type of each individual – this is mostly applicable when using the web-based application of the package. Here following coding is used:

0 – Founder

1 – Selection

2 – Reproduction

3 – Recombination

4 – Selfing

5 – DH-Production

6 – Cloning

7 – Combine

8 – Aging

9 – Split

```
> get.creating.type(population, gen=1:3)
 M1_1  M2_1  M3_1  M4_1  M5_1  M6_1  M7_1  M8_1  M9_1 M10_1 M11_1 M12_1 M13_1 M14_1 M15_1
    0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
M16_1 M17_1 M18_1 M19_1 M20_1 M21_1 M22_1 M23_1 M24_1 M25_1 M26_1 M27_1 M28_1 M29_1 M30_1
    0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

...

W31_2 W32_2 W33_2 W34_2 W35_2 W36_2 W37_2 W38_2 W39_2 W40_2 W41_2 W42_2 W43_2 W44_2 W45_2
    2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
W46_2 W47_2 W48_2 W49_2 W50_2  M1_3  M2_3  M3_3  M4_3  M5_3  M6_3  M7_3  M8_3  M9_3 M10_3
    2     2     2     2     2     1     1     1     1     1     1     1     1     1     1
```

6.11 get.cullingtime

This function extracts the time of culling of each individual – this is mostly applicable when using the web-based application of the package.

6.12 get.individual.loc

Function to derive the position in the stored population-list.

```
> get.individual.loc(population, gen=1)
      generation sex individual nr.
M1_1           1   1             1
M2_1           1   1             2
M3_1           1   1             3
M4_1           1   1             4
M5_1           1   1             5
```

6.13 get.vcf

Function to export genomic data in a vcf-file (currently using the synbreed-package but more efficient implementation including stored bp etc. is planned). Set **path** to the path you want to write to:

```
##fileformat=VCFv4.1
##filedate= 17980
##source="write.vcf of R-synbreed"
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  M1_3_set2   M10_3_set2  M2_3_set2   M3_3_set2
chr1    .   Chr1SNP1    A   G   .   PASS    .   GT  0|0 0|1 1|0 0|0 0|0 0|0 0|0 0|0 0|0 0|0
chr1    .   Chr1SNP2    A   G   .   PASS    .   GT  1|0 0|1 1|1 0|1 0|1 0|1 0|0 0|0 1|0 0|0
chr1    .   Chr1SNP3    A   G   .   PASS    .   GT  0|1 1|0 0|0 1|0 0|0 1|0 1|1 0|1 0|1 0|0
chr1    .   Chr1SNP4    A   G   .   PASS    .   GT  1|0 0|1 1|0 1|1 0|1 1|1 0|0 1|0 0|0 0|0
chr1    .   Chr1SNP5    A   G   .   PASS    .   GT  0|0 0|1 0|0 0|0 0|0 0|0 0|0 0|0 0|0 0|0
```

6.14 get.pedmap

Function to export genomic data in a ped and a map-file (PLINK format (Purcell et al. 2007)). The first of the two colums of each marker is representing the first haplotype of the individual. Set **path** to the path you want to write to:

```
1 1 0 0 0 0 C C C C C C C C A A A C C C A C A C A A C C A C A A
1 2 0 0 0 0 A A A A A A A A A A A A A C C A A A A A A C C C A A
1 3 0 0 0 0 A C A A C C C C C C A A A A A A C A C C A A A C C C
1 4 0 0 0 0 A C A A C C A C C C A A A A A C C A C C C C A A A A
1 5 0 0 0 0 A A A A A A A A A A A A A C A A A A A A A A C C C A
1 6 0 0 0 0 A A A A A C C A A A A A A A A A A C A A A A A C A C
1 7 0 0 0 0 A A A A A A C C C C C C A A A A A A A C C C C C C A
1 8 0 0 0 0 A C A C C C C A C A C C C C A A A C C C C C A C A A
1 9 0 0 0 0 C C C C A C C C C A A C A C A A A C C A A A A A C A
1 10 0 0 0 0 C A A A C C C A C C C C A A A A A A A C C C C A C
1 11 0 0 0 0 A C A A C C C C C C C A A A A C A C C C C A C A C C
```

```
1 SNP1 0 0
1 SNP2 0 0
1 SNP3 0 0
1 SNP4 0 0
1 SNP5 0 0
1 SNP6 0 0
1 SNP7 0 0
1 SNP8 0 0
1 SNP9 0 0
```

6.15 get.database

Function to merge **gen**, **database** and **cohorts** –info into a joint database. This is only needed internally – as it is the only internal *get.X() function it is still mentioned here for completeness.

```
> get.database(population, gen=1, cohorts="Selected_bulls")
         start end
[1,] 1 1     1  50
[2,] 1 2     1  50
[3,] 3 1     1  10
```

7 Importing information to the population-list

7.1 Insert.bve

To manually insert breeding values (type="bve"), true genetic values (type="bv") or phenotypes (type="pheno") use the function insert.bve. Output is a modified population list. In case new phenotypes are observed this is counted as **count** observations. In case bve are changed it is assumed that genotyping was necessary unless **count** is set to 0. This is only relevant for economic calculations.

New observations are entered in the parameter **bves** with the first column coding the individual and the others containing values for the traits:

```
> bves
     Individual Name Trait 1 Trait 2
[1,] "M1_1"          "101.5" "104.2"
[2,] "M2_1"          "102"   "98.9"
[3,] "M3_1"          "99.7"  "98.4"
[4,] "M4_1"          "98.2"  "101.2"
[5,] "M5_1"          "103.8" "101.1"
> population <- insert.bve(population, bves)
```

Structure of individual names is the same in all export functions (sex ["M"/"F"], individual number [1,2,…], and generation [1,2,…]). It is recommend to just use the names as they are exported via *get.geno()* ect..

8 Data structure of the population list

All information regarding the breeding program are stored in a population list (R-object: list) which is modified by each run of *breeding.diploid()* and *creating.diploid()*. The population list contains matrices, inside of lists, inside of lists, inside of lists, inside of lists, inside of lists, inside of lists (you get the point!) – when understanding the structure behind it is actually not that bad, luckily you do not have to understand the structure behind for most applications since you can use exporting function discussed in section 6.

The list contains two major parts - $info ((or [[1]])) and $breeding ((or[[2]]):

8.1 $info

$info contains all general information concerning genetic architecture, size of the program and internal information needed to perform the simulations. Each entry is names according to what it is supposed to contain.

schlather.slot1	Internal variable for miraculix (cre: M. Schlather)
chromosome	Number of chromosomes in the population list
snp	Number of SNPs per chromosome
position	Position (in Morgan) on the chromosome for each marker
snp.base	Major/Minor Allele (e.g. characters since internally 0/1 is used)
snp.position	Overall position in the genome (ongoing over chromosomes
length	Length of each chromosome
length.total	Cumulative length of chromosomes
func	It's just FALSE – placeholder for later
size	Size of each group (generation/sex)
bve	Coding if breeding values are simulated
bv.calculated	Coding if breeding values are calculated for the founders (will be after first run of *breeding.diploid()*)
breeding.totals	Recap of each run of *breeding.diploid()* (if stored)
bve.data	Recap of each breeding value estimation (if stored)
bve.nr	Number of traits (with QTLs behind) to consider

bv.random	Coding which traits have underlying QTLs behind
bv.random.variance	Genetic variance for traits with no QTLs behind
snps.equidistant	Are SNPs equidistant on every chromosome (speed up!)
origin.gen	List of founding generations (with stored haplotypes)
cumsnp	Cumulative sum of SNP number (just to save computational time)
bp	Physical position of each marker (bp)
snp.name	Name of each marker
next.animal	ID of the next individual to generate
bve.mult.factor	(bv) * this
bve.poly.factor	(bv) ^ this
base.bv	This + QTL_effects
bv.calc	Number of total traits (including those with no QTL behind)
real.bv.length	Traits with (additive/multiplicative/dice-effects)
sex	Sex of the founders added in *creation.diploid*
real.bv.add	Lists with an overview of all single marker QTLs for each trait
real.bv.mult	Lists with an overview of all two marker QTLs for each trait
real.bv.dice	Lists with an overview of all three+ marker QTLs for each trait
pheno.correlation	Correlation matrix of the environmental variance between traits
bv.correlation	Correlation matrix of the genetic variance between traits (only for non-QTL traits)
miraculix	Coding if miraculix was used to generate the data – only miraculix users will be able to work with those population lists
cohorts	List of all cohorts with name and position in the population list
effect.p.add	Markers involved as QTL in the single QTL proportion
effect.p.mult1	Markers involved as QTL in the two-SNP QTL proportion
effect.p.mult2	Markers involved as QTL in the two-SNP QTL proportion
effect.p	Markers involved as QTL in any trait
store.effect.freq	Frequency of each marker in each generation
last.sigma.e.database	Database to derive the last used environmental variance
last.sigma.e.value	Last used environmental variance
last.sigma.e.heritabilty	Heritability assumed to derive last used environmental variance
comp.times	Computation times needed in each use of *breeding.diploid()* (if stored)
comp.times.bve	Computation times needed in the breeding value estimation in each use of *breeding.diploid()* (if stored)
comp.times.generation	Computation times needed in for the generation of new individuals in each use of *breeding.diploid()* (if stored)
Culling.stats	Information on the culling reason of each individual (mostly releveant for the web-interface)

8.2 $breeding

$breeding contains all relevant information concerning the individuals of the breeding scheme. For efficiency purposes a lot of this is internally coded or computed on-the-fly.

Individuals are sorted according to generation, sex and individual number. In case data has to be stored for both male and female (or father/mother) there will be two entries with the first one being the male (Have to talk with the equality commissioner about that!).

$breeding[[generation]][[sex]][[individual nr.]] ((or [[2]][[generation]][[sex]][[individual nr.]]))

8.2.1 Storage per generation

$breeding[[generation]][[3,4]]	Estimated breeding values of males (3) and females (4)
$breeding[[generation]][[5,6]]	Class of males (5) and females (6)
$breeding[[generation]][[7,8]]	Underlying "true" genetic values of males (7) and females (8)
$breeding[[generation]][[9,10]]	Observed phenotypes for males (9) and females (10)
$breeding[[generation]][[11,12]]	Time point of generation for male (11) and females (12)
$breeding[[generation]][[13,14]]	Creating type of generation for males(13) and females (13) This is only relevant for the web-based application
$breeding[[generation]][[15,16]]	Individual IDs for male (15) and females (16)
$breeding[[generation]][[17,18]]	Time of culling for male(17) and female (18) individuals
$breeding[[generation]][[19,20]]	Reliability estimated for male (19) and females (20)
$breeding[[generation]][[21,22]]	Last applied selection index (mostly relevant for complex selection indices like (Miesenberger 1997)

```
> str(population$breeding[[1]])
 .. [list output truncated]
$ : num [1, 1:100] 46 48.9 31.5 35.7 49.6 ...
$ : num [1, 1:1000] 67.6 57.4 68.8 53 39.6 ...
$ : int [1:100] 0 0 0 0 0 0 0 0 0 0 ...
$ : int [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
$ : num [1, 1:100] 33.4 47.5 31.9 59.1 38 ...
$ : num [1, 1:1000] 77.79 54.65 63.42 54.49 7.45 ...
$ : num [1, 1:100] -22.23 67.24 -9.34 -16.06 101.5 ...
$ : num [1, 1:1000] 80.4 48.1 96.2 37.1 30.5 ...
```

8.2.2 Storage per individual

$breeding[[generation]][[sex]][[individual nr.]]…

[[1,2]]	Points of recombination on the first (1) and second (2) chromosome set
[[3,4]]	Points of mutations
[[5,6]]	Efficiently stored origins of segments between two points of recombination. Decoding using *decodeOrigins()* (miraculix) / *decodeOriginsR()* (else). Output in *get.recombi()* is automatically decoded
[[7,8]]	Father / Mother
[[9,10]]	Efficiently stored haplotypes (if it is a founder – else empty)
[[11,12]]	Storage of duplications (long not used!)

[[13,14]]	Storage of history of recombinations
[[15]]	How often a phenotype was generated for the individual
[[16]]	Is the individual genotyped
[[17]]	True breeding value before gene editing
[[18]]	Generation of death and previous class
[[19]]	Share of the genetic material of the grandfather of the father inherited
[[20]]	Share of the genetic material of the grandfather of the mother inherited
[[21]]	List of all individuals with the same id.

```
> str(population$breeding[[1]][[1]][[1]])
List of 21
 $ : num [1:6] 0 0.2 0.4 0.6 0.8 1
 $ : num [1:6] 0 0.2 0.4 0.6 0.8 1
 $ : NULL
 $ : NULL
 $ : int [1:5] 0 0 0 0 0
 $ : int [1:5] 1 1 1 1 1
 $ : num [1:3] 1 1 1
 $ : num [1:3] 1 1 1
 $ :Haplotype information at 1000 loci.
Attributes are a List of 3
 $ information: int [1:22] 0 1000 1 530385344 0 530385344 0 0 0 NA ...
 $ method     : int 8
 $ class      : chr "haplomatrix"
 $ : chr "Placeholder_Pointer_Martin"
 $ : NULL
 $ : NULL
 $ : NULL
 $ : NULL
 $ : num [1:21] 0 0 0 0 0 0 0 0 0 0 ...
 $ : int 0
 $ : NULL
 $ : NULL
 $ : NULL
 $ : NULL
 $ : num [1:2, 1:3] 1 2 1 1 1 63
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:3] "generation" "sex" ""
```

9 Utility functions

Since the inclusion of miraculix, I did not need many utility functions any more – older version of function to show development of allele frequencies and others still exists but need modifications. Just tell me if you wish to have a certain function to help you with the package.

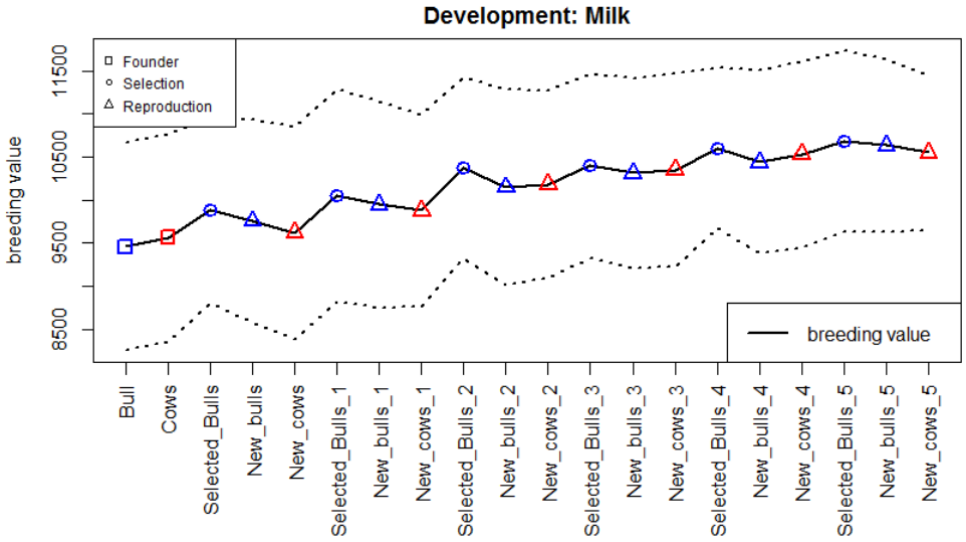
9.1 bv.development

This function will generate a plot showing the development of breeding values and phenotypes over generations. 95% confidence bands are included in a dotted line:

Which groups to display is selected via the parameters **gen**, **database** & **cohorts**. In case the user interface was used to generate the population list set **json** to TRUE to automatically display all selected cohorts. Confidence bands are drawn for “bv” (1), “bve” (2) & “pheno”(3) – to change the quantile use the parameter **quantile** (default: 0.95), to exclude selected the for with to draw a confidence band via the parameter **confidence** (default: c(1,2,3)). Groups with only zeros are ignored on default – if you want lines to be included for all selected cohorts set **ignore.zero** to FALSE.

To display the time point, the creating type, the sex, and cohort name set **display.time.point**, **display.creating.type**, **display.cohort.name**, **display.sex** to TRUE. In case the generating interface between groups is highly heterozygous it might be useful to use **equal.spacing** between displayed cohorts. To not display the line displaying a long-term trend of the breeding values set **display.line** to FALSE.

```
population <- json.simulation(total = ex_json)
bv.development(population, json=TRUE, bvrow=1, confidence = 1, development = 1,
               display.creating.type = TRUE, display.sex = TRUE,
               display.cohort.name = TRUE)
```

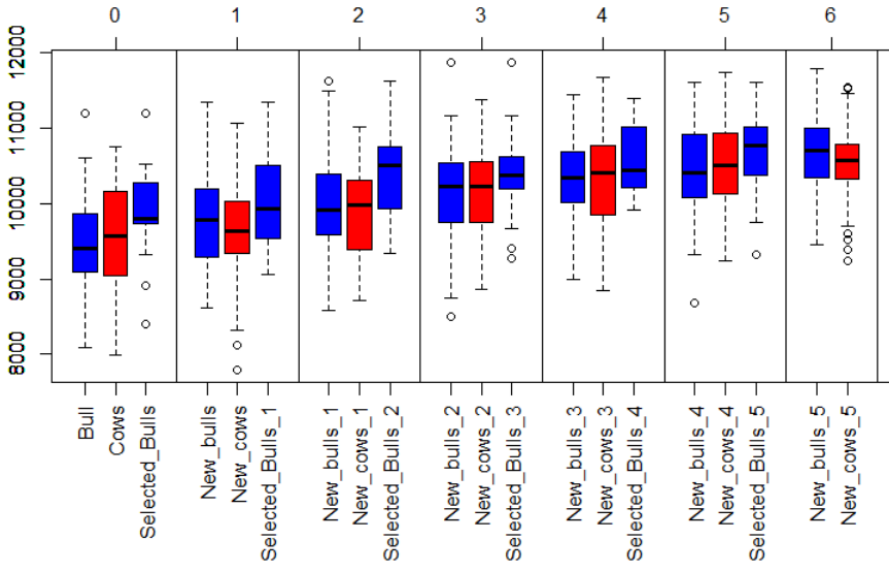


9.2 bv.development.box

This function will generate a plot displaying the development of breeding values with a boxplot for each selected **gen**, **database**, **cohorts**. In case the user-interface was used to generate the population set **json** to TRUE to automatically display all selected cohorts. To only display a subset of trait set **bvrow** to those traits. To display phenotypes or breeding value estimations for the individuals instead of breeding values, set **display** to “pheno” / “bve”.

In case the user interface was used to generate the population, one can display which cohorts where generated by which other cohort (via selection or reproduction) by setting **display.selection** and **display.reproduction** to TRUE.

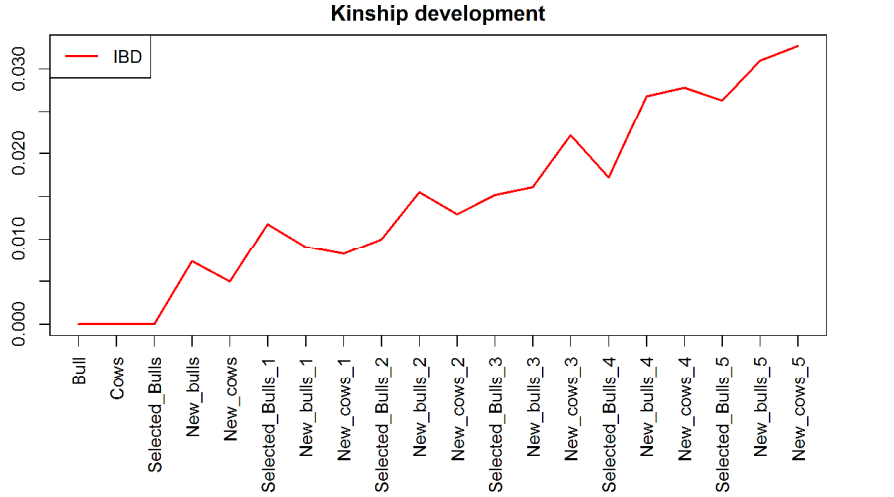
```
population <- json.simulation(total = ex_json)
bv.development.box(population, json = TRUE, bvrow = 1)
```



9.3 Kinship.development

Function to display the development of kinship over different gen, database, cohorts. Internally kinship.emp.fast is used and same optional parameters can be used to improve computation time / accuracy.

```
population <- json.simulation(total = ex_json)
kinship.development(population, json = TRUE, display.cohort.name = TRUE)
```



9.4 Kinship.emp / kinship.emp.fast

These functions can be used to derive empirical kinship between a set of individuals. Either directly supply a list containing all stored information for the respective individuals via the parameter **animals** or selected them by usage of **gen**, **database**, **cohorts**. In that case the **population** list needs to also be provided.

On default this call lead to all pairwise relations being evaluated. For a quick evaluation use *kinship.emp.fast()* and provide the total number of pairwise relationships (**ibd.obs**) and relations with the individual with itself (**hbd.obs**).

9.5 Kinship.exp

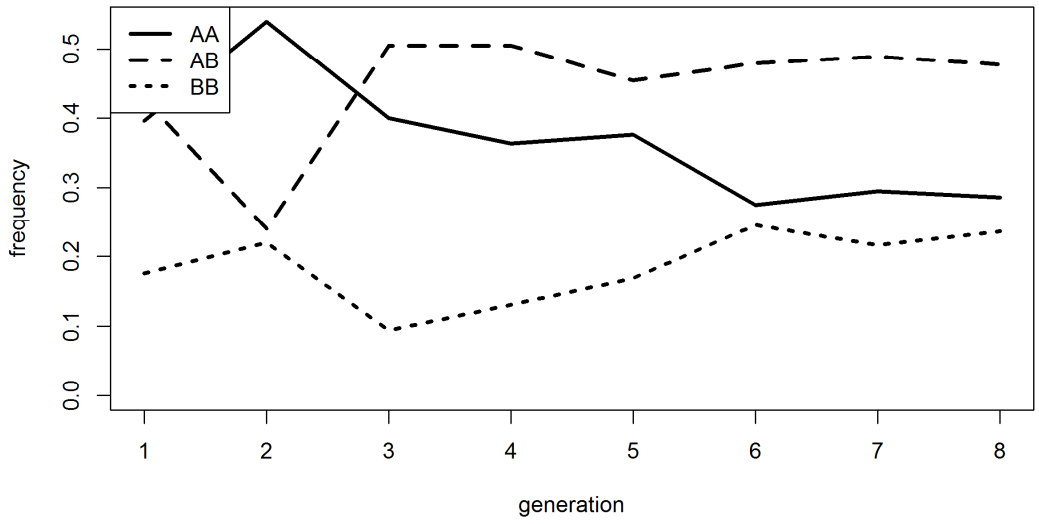
This function can be used to derive the expected kinship between individuals. To how many generations are used back use **prev.gen**. On default it's assumed all individuals before are unrelated. Alternatively, one can provide a **kinship.matrix** the individuals of the first generation via **first.individuals**. It should be noted that there are more efficient ways to derive a pedigree matrix than this – alternatively one can export the pedigree via *get.pedigree()* and use that as input for breedR or tools outside of R.

9.6 analyze.population

With this function, one can analyze the allele frequency of a specific marker over time. Select the marker to analyze via parameters **chromosome** & **snp**. To selected with generations to compare use **gen**, **database**, **cohorts**.

```
population <- json.simulation(total = ex_json)
analyze.population(population,5,2, gen=1:8)
```

```
> analyze.population(population,5,2, gen=1:8)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]  198   27  220  200  207  151  162  143
[2,]  214   12  278  278  250  264  269  239
[3,]   88   11   52   72   93  135  119  118
```



9.7 new.base.generation

With rising number of generations the number of points of recombinations to store is increasing. For efficient storing it can make sense to compute and store haplotypes for a later generation and use those individuals as a new founder generation. For this use *new.base.generation()* and select the new base generation via the parameter **base.gen**. To further reduce memory needs and computation time you can additionally delete data of previous generations via **delete.previous.gen**, **delete.breeding.totals** and **delete.bve.data**.

9.8 creating.trait

With this function one can generate additional traits for the base population without the need to add genetic datasets. Functionality is the same as *creating.diploid()* otherwise.

9.9 ensembl.map

Via this functions genetic maps provided in Ensembl (https://www.ensembl.org/index.html & http://plants.ensembl.org/index.html) can be imported. Internally the package biomaRt is used – for guidelines on how to install this package we refer to https://bioconductor.org/packages/release/bioc/html/biomaRt.html.

Naming of parameters is orientated according to the biomaRt package. Set **dataset** to the dataset you want to access (e.g. for cattle-SNPs: “btaurus_snp”) – for a list of possible datasets run this function with **export.datasets** set to TRUE.

To import a subset of all markers use the parameter filter and **filter.values**. To limited the markers to a specific SNP-chip just set **filter.values** to the name of the chip (e.g. **filter.values**=”Illumina BovineSNP50 BeadChip”). Names of potential filters for a dataset can be exported by setting **export.filters** to TRUE.

For us the direct export in R via Ensembl was painfully slow and the package did not always do what we intended it to do. Therefore, we are providing exemplary map files for most common species in the associated R-package MoBPSmaps. For a list of map that are already included in MoBPS and the associated data package MoBPS_maps we refer to section 12.

```
cattle_map <- ensembl.map(dataset="btaurus_snp", filter.values="Illumina BovineSNP50 BeadChip")
> cattle_map[1:10,]
      Chromosome SNP-ID        bp        M  freq
 [1,] "1"        "rs42778024"  "435963"  NA NA
 [2,] "1"        "rs41609588"  "776231"  NA NA
 [3,] "1"        "rs108982244" "907810"  NA NA
 [4,] "1"        "rs29026917"  "1073496" NA NA
 [5,] "1"        "rs29015852"  "1150763" NA NA
 [6,] "1"        "rs108981857" "1566539" NA NA
 [7,] "1"        "rs108994381" "1695632" NA NA
 [8,] "1"        "rs41635940"  "1929664" NA NA
 [9,] "1"        "rs41255293"  "2082139" NA NA
[10,] "1"        "rs41580909"  "2235492" NA NA
```

9.10 compute.costs

To calculate the costs of the currently simulated breeding program use the function *compute.costs()*. Currently implemented cost factors include the following:

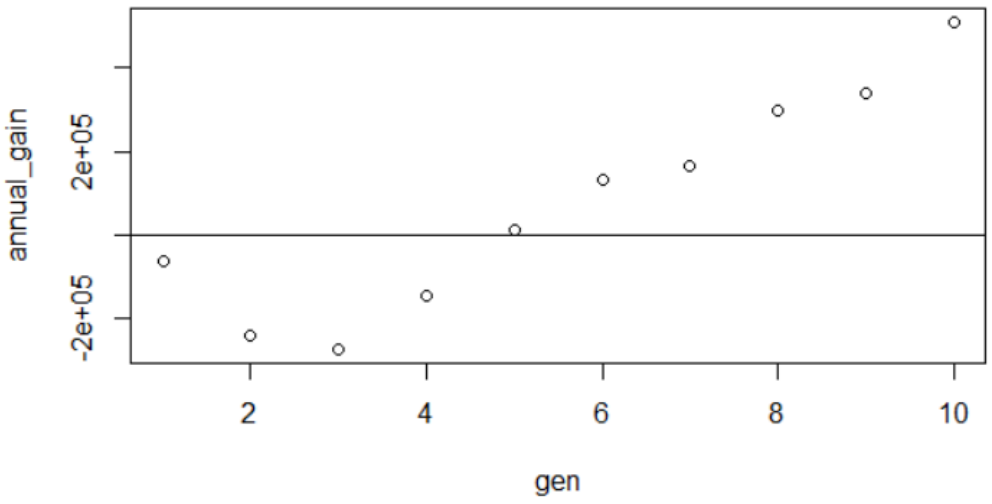
Cost factor	MoBPS -Parameter	Default
Phenotyping	phenotyping.costs	10
Genotyping	genotyping.costs	100
Housing/Field costs	Housing.costs	0
Fixed costs	fix.costs	0
Annual costs	fix.costs.annual	0
Profit per BV	profit.per.bv	1

Note that all default settings are basically chosen at random and should be modified when analyzing a real breeding program. In case costs/gains between sexes are different, use a vector. To separate between generations use a matrix with each row coding costs/gains per generation.

To only calculate the resulting costs of some generations/cohorts use database/gen.

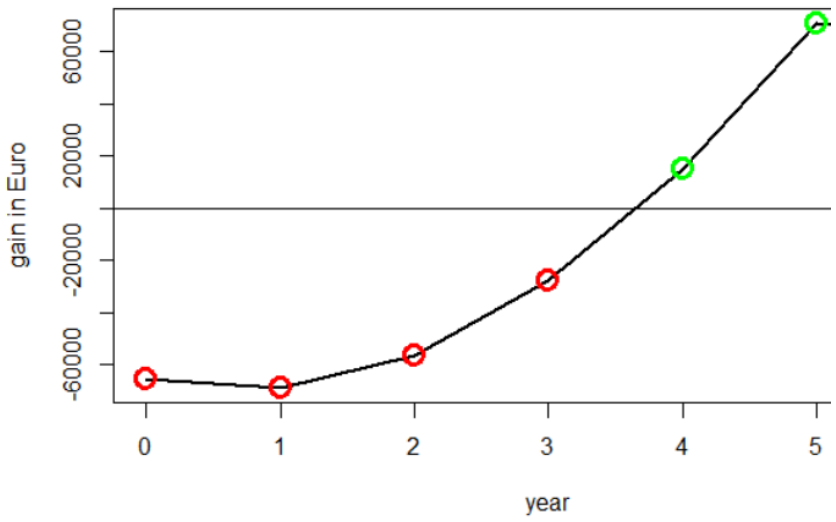
To model an interest rate set **interest.rate** (default: 1 – meaning i = 0%. We here assume inputs of the form 1 + i) – with costs/gain changed according to a base generation (**base.gen** – default: 1)

```
> compute.costs(population,gen=1:10)
 [1]  -63014.02 -242252.85 -274059.21 -144566.68   12944.92  132774.58  165916.47  299725.18  339646.66  510074.31
```



9.11 compute.costs.cohorts

The functionally of *compute.costs.cohorts()* is similar to *compute.costs()* with the added benefit of usability with input parameters provided in our user-interface. Input parameters include **phenotyping.costs**, **genotyping.costs**, **housing.costs**, **fix.costs**, **fix.costs.annual**, **profit.per.bv**, **interest.rate**. In case the user interface is provided considered groups (**gen**, **database**, **cohorts**) are automatically assigned to their time point of creation for discounting.



9.12 summary

The population list generated via *creating.diploid()* and *breeding.diploid()* is of class "population". Application of the generic function summary leads to an overview of the population list including the number of individuals, cohorts, structure of the genome and traits:

```
> summary(pop)
Population size:
Total: 100 Individuals
Of which 50 are male and 50 are female.
There are 2 generations
and 4 unique cohorts.

Genome Info:
There are 5 unique chromosomes.
In total there are 10000 SNPs.
The genome has a total length of 10 Morgan.
No physical positions are stored.

Trait Info:
There is 1 modelled trait.
The trait has underlying QTL
The trait is named: Trait 1
```

9.13 pedmap.to.phasedbeaglevcf

The standard input of MoBPS are haplotypes (not genotypes!). In case of using own genomic data it is highly encourages to perform genomic phased before using the dataset as an input. In this function a routine pipeline to generate a phased dataset is executed. In this pipeline BEAGLE 5.0 (https://faculty.washington.edu/browning/beagle/beagle.html) and PLINK 1.9 (https://www.cog-genomics.org/plink/1.9/) are used. Additional file are generated in a selected directory. Path for all three have to be provided in **beagle_jar**, **plink_dir**, **db_dir**. Input can either be a dataset in PLINK format (**ped_path**, **map_path**) or a vcf-file (**vcf_path**). Defaults are all chosen to work on the

webserver for our web-interface (Chapter 14). Phasing can also be directly performed in the web-interface.

10 Memory and computation times

Critical parts of MoBPS concerning memory requirements and computation times can be performed using the associated R-package miraculix. By using SSE2 operations and bit-wise storing computation speed can be massively increased leading to about 10 times faster matrix multiplications than the regular R implementation while needing only 1/16 of the regularly needed memory.

To speed up commutation of the breeding value estimation one can use multiple cores by the usage of **miraculix.cores** (default: 1) or in case miraculix is not active **ncore**. To parallelize generation of new individuals set **parallel.generation** to TRUE and set the number of cores used via **ncore.generation**. This will only lead to significant improvement in computation time for the generation of a lot of individuals. Even when using a single core ~1'000 individuals are generated per second. The packages doParallel (Microsoft Corporation and Steve Weston 2018) and doRNG (Renaud Gaujoux 2018) are used for parallelization in R.

11 List of input parameters in breeding.diploid()

For a description of each parameter we refer to the use of the help function in R (?*breeding.diploid*) and/or other sections of this Guidelines.

Parameter	**Default**	**options**
population	NULL	A previous population list
mutation.rate	10^-5	Value between 0 and 1
remutation.rate	10^-5	Value between 0 and 1
recombination.rate	1	Any positive numeric
selection.m	"random"	"function"
selection.f	NULL (selection.m)	"random", "function"
new.selection.calculation	TRUE	FALSE
selection.function.matrix	NULL	Don't touch – will be removed
selection.size	C(0,0)	Vector with two non-negative values
breeding.size	0	Positive number // 2 element vector (male/female)
breeding.sex	0.5	Value between 0 and 1
breeding.sex.random	FALSE	TRUE
class.m	0	Vector with all classes to consider
class.f	NULL (class.m)	Vector with all classes to consider
add.gen	0 (will lead to added generation)	Value between 1 and number of generations

recom.f.indicator	NULL	Not necessary (use modified marker position instead!)
recom.f.polynom	NULL	Not necessary (use modified marker position instead!)
duplication.rate	0	
duplication.length	0.01	Duplication modelling needs changes!
duplication.recombination	1	
same.sex.active	FALSE	TRUE
new.class	0	Numeric value (ideally positive integer; -1 is reserved for dead individuals)
bve	FALSE	TRUE
bve.direct.est	TRUE	FALSE
bve.gen	NULL	1:3
bve.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
bve.cohorts	NULL	c("Founder_M", "F1")
bve.avoid.duplicates	TRUE	FALSE
report.accuracy	TRUE	FALSE
sigma.e	NULL	Numeric value above 0
sigma.s	100	Numeric value above 0
new.bv.observation	NULL	"all" for all individuals "non_obs" for all previously not observed "non_obs_m" for all previously not observed male individuals "non_obs_f" for all previously not observed female individuals
new.bv.observation.gen	NULL	1:3
new.bv.observation.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
new.bv.observation.cohorts	NULL	c("Founder_M", "F1")
new.bv.child	"zero"	"mean", "obs"
computation.A	"vanRaden"	"kinship", "CE", "CM", "non_stand"
depth.pedigree	7	Positive Integer
delete.haplotypes	NULL	Vector of all generations to delete (natural number)
delete.individuals	NULL	Vector of all generations to delete (natural number)

fixed.breeding	NULL	matrix with each row containing (gen1,sex1,nr1, gen2,sex2,nr2, sex.probability) with 1 being father, 2 being mother)
fixed.breeding.best	NULL	matrix with each row containing (sex1, nr1, sex2, nr2, sex.probability) chosen from the group of selected individuals
max.offspring	C(Inf,Inf)	vector with two natural numbers (first male, second female)
store.breeding.totals	FALSE	TRUE
forecast.sigma.s	TRUE	FALSE
multiple.bve	“add”	“ranking”
multiple.bve.weights.m	1	Any weights – use a vector with length equal to number of traits
multiple.bve.weights.f	NULL (multiple.bve.weights.m)	Any weights – use a vector with length equal to number of traits
store.bve.data	FALSE	TRUE
fixed.assignment	FALSE	“bestworst”, “worstbest”
reduce.group	NULL	Per row: Generation, Sex, Individuals to survive, class of individuals
reduce.group.selection	“random”	“function”
selection.criteria	c(TRUE,TRUE)	C(FALSE/TRUE,FALSE/TRUE)
same.sex.sex	0.5	Numeric value between 0 and 1
same.sex.selfing	TRUE	FALSE
selfing.mating	FALSE	TRUE
selfing.sex	0.5	Numeric value between 0 and 1
praeimplantation	NULL	No use currently recommended
sigma.e.gen	NULL	1:3
sigma.e.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
sigma.e.cohorts	NULL	c(“Founder_M“, “F1“)
heritability	NULL	Numeric value between 0 and 1
multiple.bve.scale.m	FALSE	TRUE
multiple.bve.scale.f	NULL (multiple.bve.scale.m)	TRUE
use.last.sigma.e	FALSE	TRUE
save.recombination.history	FALSE	TRUE
martini.selection	FALSE	TRUE

BGLR.bve	FALSE	TRUE
BGLR.model	“RKHS”	“BRR”, “BL”, “BayesA”, “BayesB”, “BayesC”
BGLR.burnin	500	natural number
BGLR.iteration	5000	natural number
BGLR.save	“RKHS”	any path you want
BGLR.save.random	FALSE	TRUE
BGLR.print	FALSE	TRUE
copy.individual	FALSE	TRUE
copy.individual.keep.bve	TRUE	FALSE
dh.mating	FALSE	TRUE
dh.sex	0.5	Numeric value between 0 and 1
offspring.bve.parents.gen	NULL	1:3
offspring.bve.parents.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
offspring.bve.parents.cohorts	NULL	c(“Founder_M“, “F1“)
offspring.bve.offspring.gen	NULL	1:3
offspring.bve.offspring.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
offspring.bve.offspring.cohorts	NULL	c(“Founder_M“, “F1“)
bve.parent.mean	FALSE	TRUE
bve.grandparent.mean	FALSE	TRUE
bve.mean.between	“bvepheno”	„bve“, „pheno“, „bv“
n.observation	1	Natural number
share.genotyped	1	Numeric value between 0 and 1
added.genotyped	0	Numeric value between 0 and 1
remove.non.genotyped	TRUE	FALSE
Singlestep.active	FALSE	TRUE
bve.0isNA	TRUE	FALSE
phenotype.bv	FALSE	TRUE
standardize.bv	FALSE	TRUE
standardize.bv.level	100	Numeric value
standardize.bv.gen	1	Natural number <= generation number
delete.same.origin	FALSE	TRUE
remove.effect.position	FALSE	TRUE
estimate.u	FALSE	TRUE
fast.uhat	TRUE	FALSE

new.phenotyp.correlation	NULL	Positive definite matrix
new.breeding.correlation	NULL	Positive definite matrix
recalculate.bv.var.correlation	FALSE	TRUE
new.bv.random.correlated	TRUE	FALSE
estimate.add.gen.var	FALSE	TRUE
estimate.pheno.var	FALSE	TRUE
selection.m.gen	NULL	1:3
selection.f.gen	NULL	1:3
selection.m.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
selection.f.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
selection.m.cohorts	NULL	c("Founder_M", "F1")
selection.f.cohorts	NULL	c("Founder_M", "F1")
best1.from.group	NULL	Matrix with one group per row
best2.from.group	NULL	Matrix with one group per row
best1.from.cohort	NULL	Vector containing names of cohorts
best2.from.cohort	NULL	Vector containing names of cohorts
Reduced.selection.panel.m	NULL	Vector containing numeric values
Reduced.selection.panel.f	NULL	Vector containing numeric values
store.comp.times	TRUE	FALSE
store.comp.times.bve	TRUE	FALSE
special.comb	FALSE	Part of martini selection – do not use!
max.auswahl	Inf	Part of martini selection – do not use!
predict.effects	FALSE	Part of martini selection – do not use!
SNP.density	10	Part of martini selection – do not use!
use.effect.markers	FALSE	Part of martini selection – do not use!
use.effect.combination	FALSE	Part of martini selection – do not use!
import.position.calculation	NULL	Function f(cm_position) = Last previous SNP

special.comb.add	FALSE	Part of martini selection – dont use!
ogc	FALSE	TRUE
ogc.cAc	NA (minimal gain in inbreeding)	Numeric value between 0 and 1
computation.A.ogc	“kinship”	“vanRaden” (see computation.A)
depth.pedigree.ogc	7	Positive Integer
emmreml.bve	FALSE	TRUE
nr.edits	0	any natural number
gene.editing	FALSE	TRUE
gene.editing.offspring	FALSE	TRUE
gene.editing.best	FALSE	TRUE
gene.editing.offspring.sex	c(TRUE,TRUE)	Vector with two boole variables
gene.editing.best.sex	c(TRUE,TRUE)	vector with two boole variables
gwas.u	FALSE	TRUE
approx.residuals	TRUE	FALSE
sequenceZ	FALSE	TRUE
maxZ	5000	Any natural number
maxZtotal	0	Any natural number
gwas.gen	NULL	1:3
gwas.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
gwas.cohorts	NULL	c(“Founder_M“, “F1“)
delete.sex	c(1,2)	1 (male), 2 (female)
gwas.group.standard	FALSE	TRUE
y.gwas.used	“pheno”	“bv“, “bve“
culling.cohort	NULL	Any cohort name
culling.time	Inf	Numeric value
culling.name	“Not_named”	Any character string
culling.bv1	100	numeric value
culling.share1	0	Probability between 0 and 1
culling.bv2	110	numeric value
culling.share2	0	Probability between 0 and 1
culling.index	0	Any weights – use a vector with length equal to number of traits, “lastindex”
gen.architecture.m	0	Natural number (select one of the previously stored architectures)

gen.architecture.f	NULL (gen.architecture.m)	Natural number (select one of the previously stored architectures)
ncore	1	Natural number
Z.integer	FALSE	TRUE
store.effect.freq	TRUE	FALSE
backend	“doParallel”	„doMPI“
randomSeed	NULL	natural number
randomSeed.generation	NULL	Natural number
Rprof	FALSE	TRUE
miraculix	FALSE	TRUE (automatically activated when miraculix is used is *creating.diploid()*)
miraculix.mult	NULL (leading to FALSE)	TRUE / FALSE
fast.compiler	0	3 (For R >= 3.4 this is default in R)
miraculix.cores	1	natrual number
store.bve.parameter	FALSE	TRUE
print.error.sources	FALSE	TRUE
chol.miraculix	FALSE	TRUE
bve.insert.gen	NULL	1:3
bve.insert.database	NULL	Generation sex [1,] 1 2 [2,] 5 1
bve.insert.cohorts	NULL	c(“Founder_M“, “F1“)
best.selection.ratio.m	1	positive numeric value
best.selection.ratio.f	NULL (best.selection.ratio.m)	positive numeric value
best.selection.criteria.m	“bv”	“bve“, “pheno“
best.selection.criteria.f	NULL (best.selection.criteria.m)	“bve“, “pheno“
best.selection.manual.ratio.m	NULL	positive numeric value
best.selection.manual.ratio.f	NULL (best.selection.manual.ratio.m)	positive numeric value
bve.class	NULL (take all!)	vector containing numeric values
parallel.generation	FALSE	TRUE
ncore.generation	1	Positive numeric value
name.cohort	NULL	“Founders” or any other character string
add.class.cohorts	TRUE	FALSE
display.progress	TRUE	FALSE

ignore.best	C(0,0)	Any two element vector (first male, second female)
combine	FALSE	TRUE
repeat.mating	1	Positive numeric value
time.point	0	Positive numeric value (this will be automatically processed in the web-based-application)
creating.type	0	This is automatically stored in the web-based-application # 0 – Founder # 1 – Selection # 2 – Reproduction # 3 – Recombination # 4 – Selfing # 5 – DH-Production # 6 – Cloning # 7 – Combine # 8 – Aging # 9 – Split

12 List of input parameters in creating.diploid()

For a description of each parameter we refer to the use of the help function in R (?*creating.diploid*) and/or other sections of this Guidelines.

Parameter	**Default**	**options**
population	NULL (will lead to "random")	A previous population list
dataset	"random"	SNP-dataset (One haplotype per colum), "random", "all0", "homorandom", "allhetero"
nsnp	0	Positive integer value
nindi	0	Positive integer value
vcf	NULL	Path to a vcf-file
map	NULL	Matrix with up to 5 colums containing (chr.nr, snp.name, bp, position in Morgan, allele freq). Rest will be set to NULL/NA. For more see section 13
chr.nr	NULL (all markers on the same chromosome)	Vector containing the chromosome for each generated marker, or natural number with the number of chromosomes
bp	NULL	Vector containing the base-pair for each generated marker

snp.name	NULL	Vector containing the snp-name for each generated marker
bpcm.conversion	0	Recommendations: For human: 100.000.000 For chicken: 30.000.000
chromosome.length	NULL (will lead to 5M)	Positive numeric value
freq	"beta"	Numeric value or vector for each marker
beta.shape1	1	Positive numeric value
beta.shape2	1	Positive numeric value
sex.s	"fixed"	"random", vector containing the sex of each newly added individual.
share.genotyped	1	Numeric value between 0 and 1
genotyped.s	NULL	vector containing the sex of each newly added individual.
add.chromosome	FALSE	TRUE
generation	1	Positive integer value (no empty generations inbetween!)
class	0	Numeric value (positive integer recommended)
sex.quota	0.5	Numeric value between 0 and 1
snps.equidistant	NULL (will be TRUE if no other way to derive Morgan-position is provided)	FALSE/TRUE
change.order	FALSE	TRUE
position.scaling	FALSE	TRUE
length.before	5	Positive numeric value
length.behind	5	Positive numeric value
hom0	NULL (automatically derived)	Vector containing major allele for each generated marker.
hom1	NULL (automatically derived)	Vector containing minor allele for each generated marker.
miraculix	TRUE	FALSE
bit.storing	FALSE	TRUE (this is less efficient than miraculix but does not rely on C-code)
nbits	30	Integer value between 1 and 30
bv.total	0 (automatically set according to traits provided)	Integer value. If higher than the number of traits simulate traits based on pedigree/inbreeding rates
trait.name	NULL	Vector containing the names of the traits (e.g. "milk")
real.bv.add	NULL	List with each element containing effect matrices

real.bv.mult	NULL	List with each element containing effect matrices
real.bv.dice	NULL	List with each element containing effect lists
n.additive	0	Positive integer value
n.dominant	0	Positive integer value
n.qualitative	0	Positive integer value
n.quantitative	0	Positive integer value
var.additive.l	NULL	List containing a single numeric value or vector with variances for each trait
var.dominant.l	NULL	List containing a single numeric value or vector with variances for each trait
var.qualitative.l	NULL	List containing a single numeric value or vector with variances for each trait
var.quantitative.l	NULL	List containing a single numeric value or vector with variances for each trait
exclude.snps	NULL	Vector containing marker positions with no simulated random effects
shuffle.traits	NULL	TRUE
shuffle.cor	NULL	Correlation matrix for the traits to shuffle
replace.real.bv	FALSE	TRUE
name.cohort	NULL	Character string
skip.rest	FALSE	TRUE (INTERNAL PARAMETER!)
randomSeed	NULL	Integer value
template.chip	NULL	“cattle”, “chicken”, “pig”, “sheep”, “maize”
time.point	0	Positive numeric value (this will be automatically processed in the web-based-application)
creating.type	0	This is automatically stored in the web-based-application # 0 – Founder # 1 – Selection # 2 – Reproduction # 3 – Recombination # 4 – Selfing # 5 – DH-Production # 6 – Cloning # 7 – Combine # 8 – Aging # 9 – Split

remove.invalid.qtl	TRUE	FALSE

13 List of datasets included in the package

MoBPS does contain a variety of maps that are preimported from Ensembl since the actual import takes quite long for bigger map-files. In case you feel a certain map is missing feel free to contact us to we can add it to the tool. Maps are available in the associated R-package MoBPS_maps. Only map_chicken1, map_cattle1 and map_maize1 are included in MoBPS itself. To use a specific map use it as an input for the parameter **map** in *creating.diploid()*.

In addition to all those maps an exemplary json-file (**ex_json**) generated by a recent version of our interface is included for text use in *json.simulation()* and other function that utilize datasets generated by *json.simulation()*. Note that this file is automatically generated via the user-interface and you do not have to worry about its structure.

Dataset name	Corresponding Chip	Number of Markers	Contains: 1. Physical position 2. Morgan position 3. allele frequency		
Map_pig1	Axiom Genotyping Array	590‘318			
Map_pig2	GGP Porcine HD	63’113			
Map_pig3	GGP Porcine LD	8’624			
Map_pig4	Illumina_PorcineSNP60	55‘684			
Map_chicken1	Affymetrix Chicken600K Array	547’024			
Map_chicken2	Affymetrix Chicken600K Array (diversity subset)	293’251			
Map_chicken3	Affymetrix Chicken600K Array (50k subset)	50’000			
Map_cattle1	Illumina BovineSNP50 BeadChip	45’613			
Map_cattle2	Illumina BovineHD BeadChip	727‘605			
Map_cattle3	Illumina BovineLD BeadChip	6’600			
Map_cattle4	Genotyping chip variations	732‘645			
Map_horse1	Illumina EquineSNP50 BeadChip	51’105			
Map_sheep1	IlluminaOvineHDSNP	575‘256			
Map_sheep2	IlluminaOvineSNP50	46‘545			
Map_sheep3	Genotyping chip variants	580‘661			
Map_goat1	Illumina_GoatSNP50	55‘050			
Map_human1	Affy GeneChip500K	483‘418			
Map_human2	Illumina_1M-duo	1‘122‘013			
Map_human3	Illumina_HumanHap550	545‘902			
Map_maize1	Affymetrix Axiom Maize Genotyping Array	501’124			

Map_wheat1	Subset of a 55k chip from (Liu et al. 2018)	12'109			
Map_wheat2	Subset of a 90K chip from (Wen et al. 2017)	29'692			
Map_sorghum1	Subset of a 90k chip from (Bekele et al. 2013)	3'000			

Ex_json – the first few rows. Do not bother trying to understand it. *Json.simulation()* will do the job for you:


```
"Nodes": [
  {
    "id": "Bull",
    "Number of Individuals": "100",
    "x": -152,
    "y": -165,
    "individualsVar": "100",
    "Founder": "Yes",
    "Path": "",
    "Proportion of Male": 1,
    "BV Plot": "Yes",
    "Sex": "Male",
    "Phenotyping Class": "Default PhenoC",
    "Housing Cost Class": "Male individuals",
    "Proportion of genotyped individuals": 1,
    "label": "Bull",
    "color": "#9acef4",
    "title": "Bull: 100 Ind"
  },
```


14 User-interface

The development of the user-interface of MoBPS is a joint project of Torsten Pook, Amudha Ganesa, Ngoc-Thuy Ha, Lisa Büttgen, Johannes Geibel and Henner Simianer (All: Department of Animal Sciences, Center for Integrated Breeding Research, University of Goettingen, Goettingen, 37075, Germany). The user-interface is currently in active development and available at www.mobps.de. Note that this is still a development version and is constantly undergoing change.

There will be a separate publication for the user interface that is in preparation but this will still take a while. The user-interface will take more developing to fulfil our standards for publication! In case you have a given simulation you want to perform feel free to contact us for possible collaboration (Torsten.pook@uni-goettingen.de). When looking through the openly-available code you will find code snippets that are only relevant for the interface (*json.simulation()*). At the current state, this chapter can mostly be seen as a news page on the current stage of development.

Main goal of the user-interface is the usage of the R-package without the need of programming skills in R or knowledge of the details of the package to set up your simulation. Note that the interface will not be able to grasp the full functionality/efficiency of the R-package but the goal is to get close. Input parameters can be entered in a web-based application (java-script) – especially the breeding scheme can be entered in an intuitive way via nodes (cohorts of individuals) and edges (breeding & selection processes).

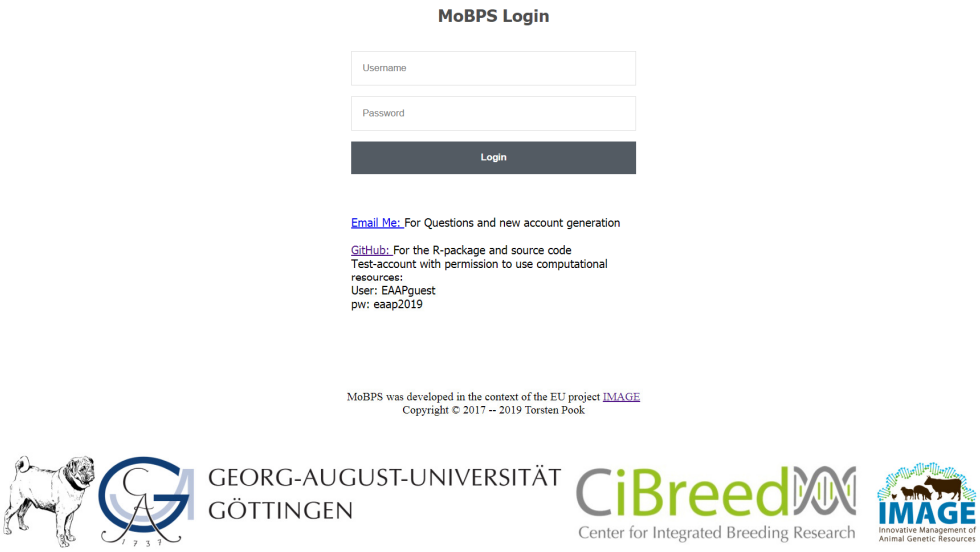
Simulations can be directly started via the web-interface with a VM server hosted from Goettingen. AT the current state we can provide computational resource for smaller generations (20 CPU, 64 GB Memory), but it is also population to download the json-file containing all information of the user interface and run the simulation via *json.simulation()* in R. The global test account (EAAPguest) has no permission to use our computing resources but scripts can still be exported. Student-Users are allowed to use up to five cores and are limited to 6 hour simulations, professional users are allowed up to 10

cores with no time limit for simulations. To generate an account or upgrade your available resources contact me (Torsten.pook@uni-goettingen.de).

To input information in regard to the breeding program in a user friendly way we provide the following module:

1. Design you Genome
2. Design your Traits
3. Multiple Subpopulations
4. Design your Selection Index
5. Reasons for Culling
6. Economic Parameters
7. Draw your Breeding Scheme
8. Analyze your Population

For most inputs we provide implemented help buttons to briefly explain what kind of input is expected. The exemplary json-script provided in the R-package would look like this (note that all advanced parameter options are deactivate to not further complicate things):



<< Ex_json >> | Save Project | Save As ... | Download Project | Delete Project | Show/Hide Info | Show/Hide Warnings

MoBPS

Load a new/existent project from your own database:

Project: Ex_json ▾ Version: (most recent) ▾

Or choose a template as a starting point:

▾

General Information

Project Name ⓘ	Ex_json
Advanced settings	☐
Species ⓘ	Cattle ▾
Time Unit ⓘ	Years ▾
Genetic Data ⓘ	◉ Use Ensembl Map ○ Upload Own Map (vcf/plink) ○ Create customized Map
Ensembl Dataset ⓘ	Illumina BovineSNP50 BeadChip ▾
Max. Number of SNPs ⓘ	10000

Phenotype Information ⓘ

Add new phenotype | Show/Hide 3 phenotypes | Show/Hide QTLs | Show/Hide phenotypic correlation | Show/Hide genetic correlation

Phenotype ⓘ	Unit ⓘ	Pheno. Mean ⓘ	Pheno. SD ⓘ	Heritability ⓘ	# polygenic loci ⓘ	Major QTL ⓘ	Value per unit (€) ⓘ	Show Cor	
Milk	liters	9300	900	0.35	1000	1	0,30	☑	X
Fat	%	3.9	0.4	0.4	1000	1	100	☑	X
Protein	%	3.4	0.3	0.38	100	0	100	☑	X

SNP for Milk	SNP ID ⓘ	bp	chromo	Effect AA	Effect AB	Effect BB	Allele Freq. (B) ⓘ	Optional Info	
1	QTL DGAT	545	14	0	-400	-800	0.2	DGAT1	X

SNP for Fat	SNP ID ⓘ	bp	chromo	Effect AA	Effect AB	Effect BB	Allele Freq. (B) ⓘ	Optional Info	
1	QTL DGAT	545	14	0	0.5	1	0.2	DGAT1	X

Upload Excel file for Correlation:

ⓘ

Datei auswählen | Keine ausgewählt | Save to Project

Residual Correlation ⓘ

	Milk	Fat	Protein
Milk	1	0.1	0.3
Fat	0.1	1	-0.2
Protein	0.3	-0.2	1

Enter Phenotypic correlation instead of residual correlation ☐

Genetic Correlation ⓘ

	Milk	Fat	Protein
Milk	1	0.3	0.4
Fat	0.3	1	0.1
Protein	0.4	0.1	1

Creating own Selection Indexes (Optional) ⓘ

Type SI name | Add new SI

SI ⓘ	Milk	Fat	Protein
Default Index	1	1	1
Milk increase	5	1	1

SI ⓘ	Selection Index (Hazel and Lush 1943, Miesenberger 1997) ⓘ	Standardization ⓘ
Default Index	☐	Per Phenotypic SD
Milk increase	☐	Per Phenotypic SD

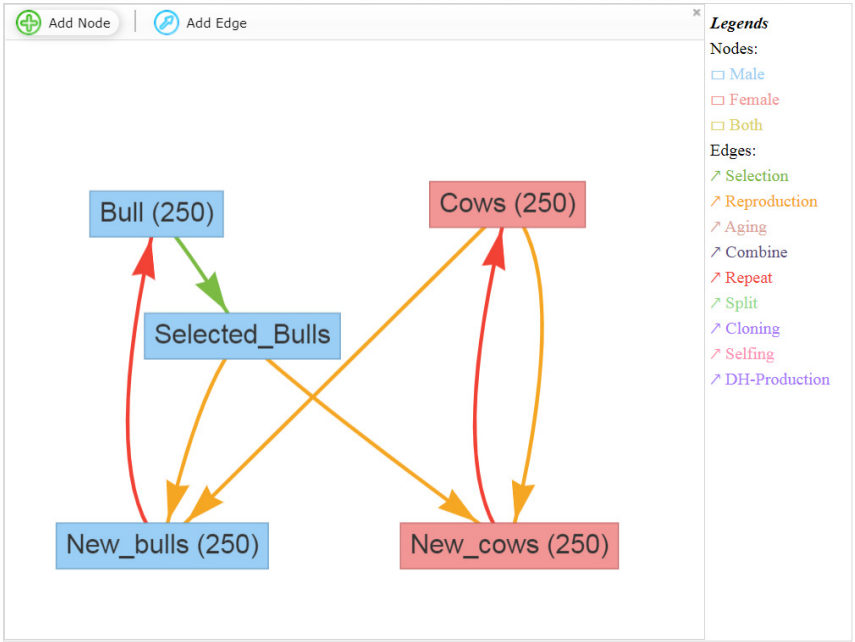
Creating own Phenotyping Classes (Optional) ⓘ

Type PhenoC name | Add new PhenoC

PhenoClass ⓘ	Phenotyping Cost	Milk	Fat	Protein	
Default PhenoC	800	1	1	1	
Male individuals	0	0	0	0	X

Breeding Scheme ⓘ

Important: Please AVOID node names with underscore and trailing numbers combined, e.g. ABC_1, DEF_2, ...



After the simulations are execute the resulting, population-list can be download in R and then be manually analyzed. Alternatively, we also provide some basic evaluation functions. In case multiple

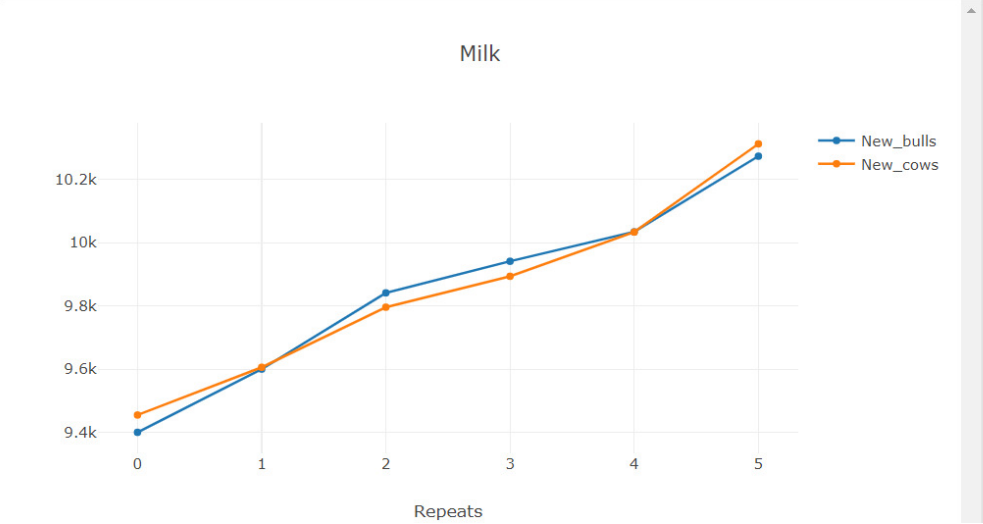
simulations are performed it is also possible to analyze average between multiple runs of the simulation:

Results: True Breeding Values

Select plotting type: By Repeats

Select cohorts (multiple selection possible): Plot Results

New_bulls (5 Repeats) × New_cows (5 Repeats) ×

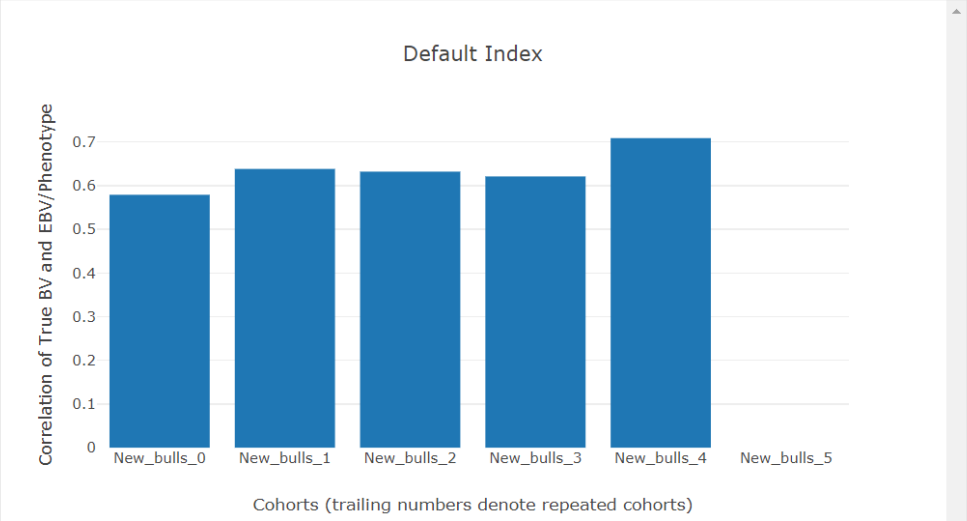


Results: Accuracy of Breeding Value Estimation

Select plotting type: By Cohorts

Select cohorts (multiple selection possible): Plot Results

New_bulls (5 Repeats) ×

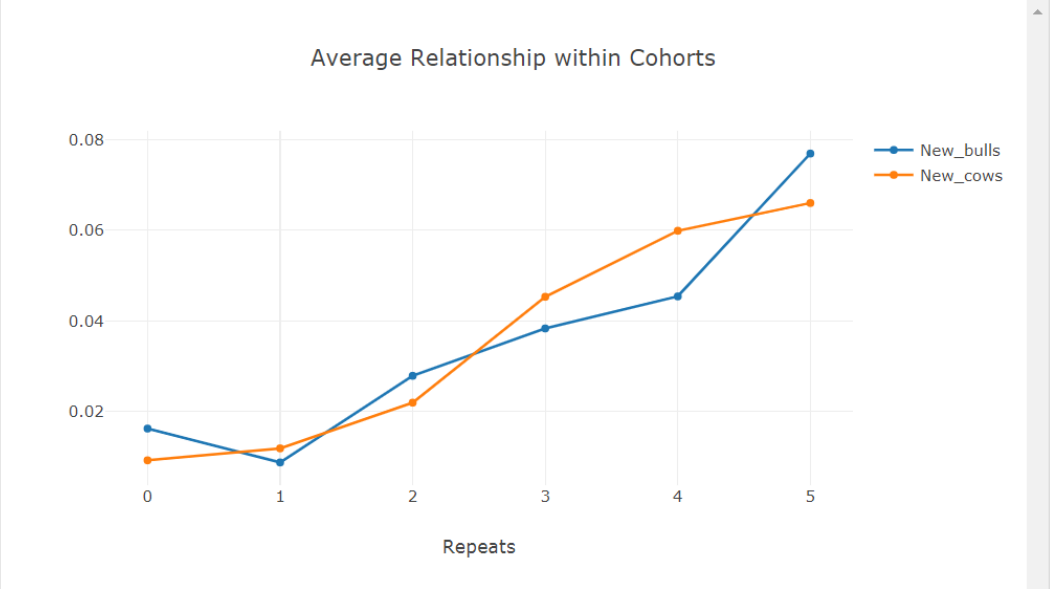


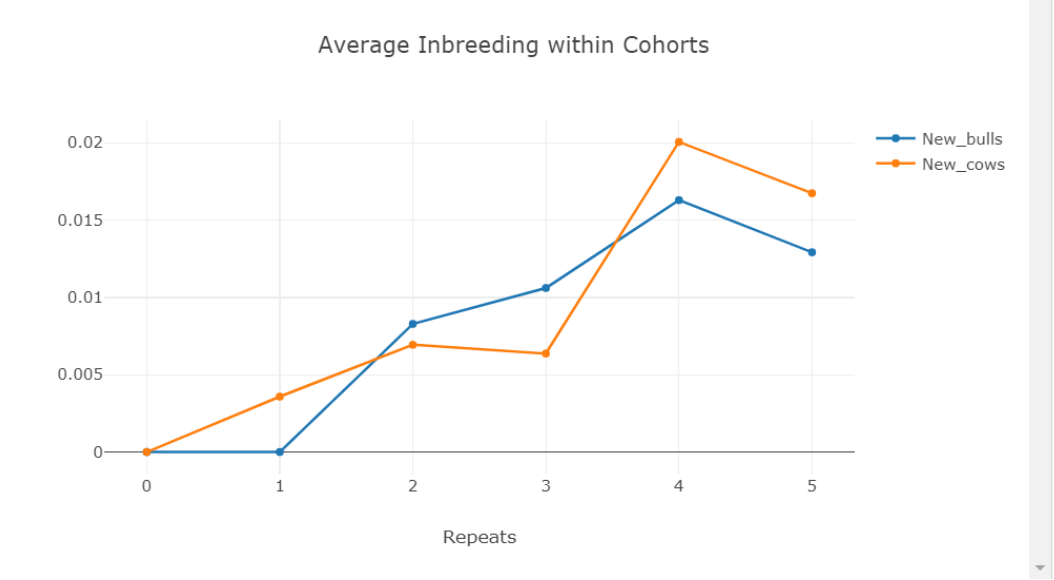
Results: Relationship and Inbreeding within Cohorts

Select plotting type: By Repeats ▾

Select cohorts (multiple selection possible): Plot Results

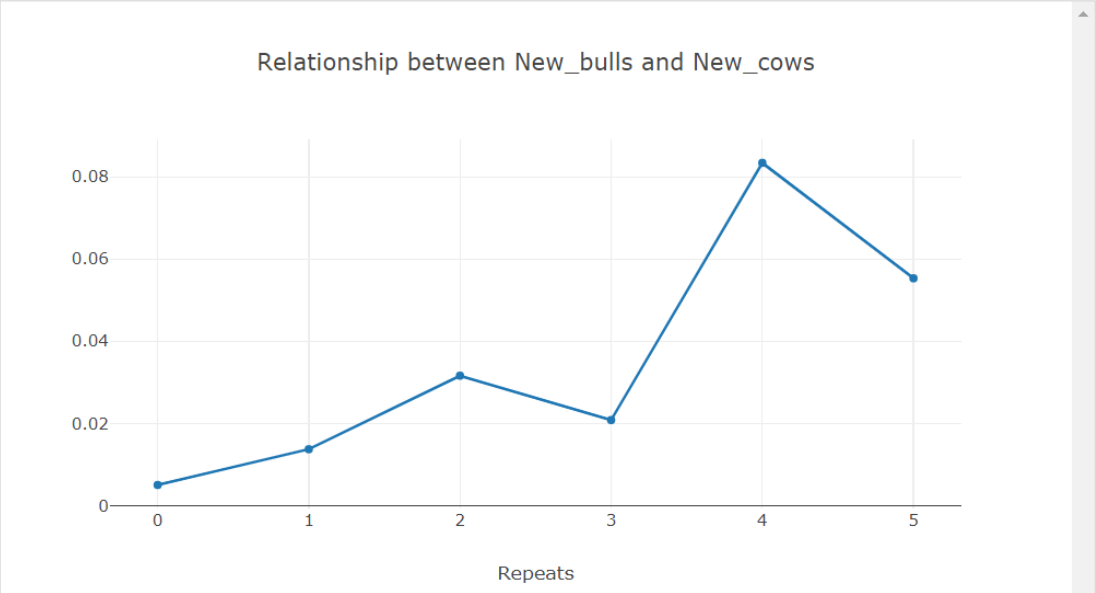
New_bulls (5 Repeats) × New_cows (5 Repeats) ×





Results: Relationship between Cohorts

Select cohort 1 : New_bulls (5 Repeats) Select cohort 2 : New_cows (5 Repeats) Plot Results

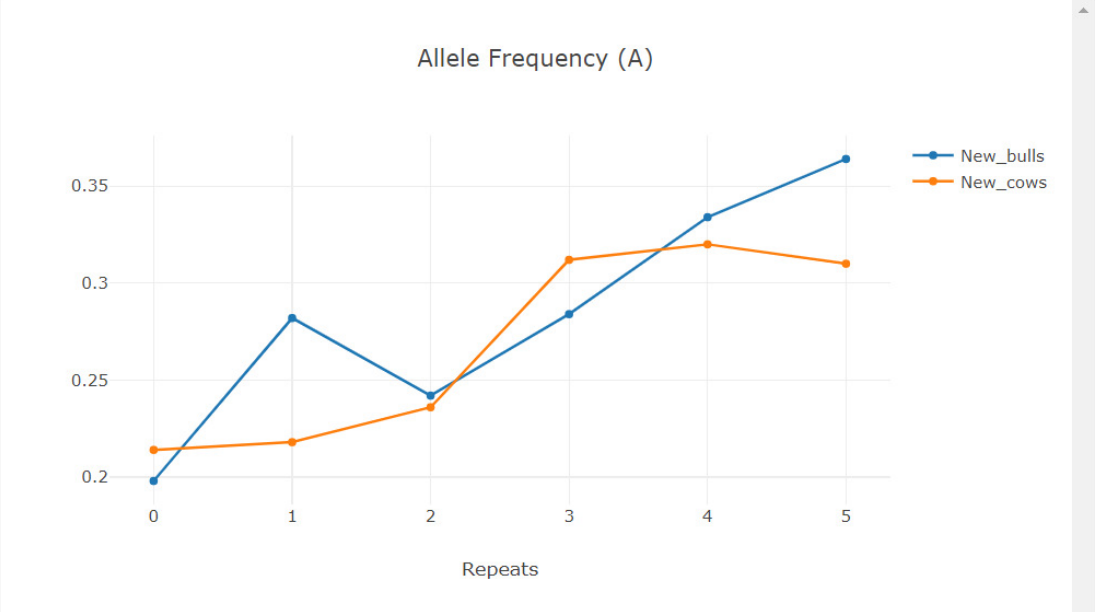


Results: Major QTLs (Allele Frequency, exp./obs. Heterozygosity)

Select Trait : Milk Select QTL : QTL DGAT1

Select Cohorts (multiple selection possible): Plot Results

New_bulls (5 Repeats) × New_cows (5 Repeats) ×



15 Commonly used word definitions

Group: Group of individuals with the same sex and belonging to the same generation

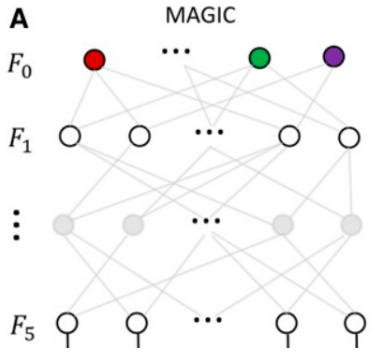
Cohorts: Group of individuals with the same sex generated in a single run of *breeding.diploid()*

Class: Auxiliary variable to classify individuals in an additional dimension (besides sex & generation)

Founder: Founder individuals are the start-point of a simulation and all individuals in the population can be traced back to the founders. Because of this only for those individuals genotype/haplotype data has to be saved.

16 Exemplary scripts

16.1 Simulation of a MAGIC population in maize



We here show how to perform an exemplary simulation of a MAGIC population in maize with a mating scheme given in (Zheng et al. 2015) – cf. adjacent Figure.

Since default settings in MoBPS are to always use the last generation anyway the needed code is quite short even without cohort mode. For the sake of completeness, we provide a cohort version for the script as well.

In term of computation time this simulation with a 15.3M genome, 31k SNPs and a total of 780 individuals took 1.6 seconds on one core of my local maschine.

Non-cohort-modus:

```
# Generation of 20 fully-homozygous founders lines
# All plants are stored as male individuals (sex=0)
population <- creating.diploid(nindi = 20, sex.quota = 0, template.chip = "maize",
                               dataset = "homorandom")

# Simulate matings between all founders.
# Each plan is involved in exactly 19 matings.
population <- breeding.diploid(population, breeding.size = c(190,0),
                               breeding.all.combination = TRUE,
                               selection.size = c(20,0), max.offspring = 19)

# Simulate matings between plants of the last generation.
# Each plant is involved in exactly 2 matings.

population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = 2)
population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = 2)
population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = 2)
```

Cohort-modus:

```
# Generation of 20 fully-homozygous founders lines
# All plants are stored as male individuals (sex=0)
population <- creating.diploid(nindi = 20, sex.quota = 0, template.chip = "maize",
                               dataset = "homorandom", name.cohort = "F0")
```

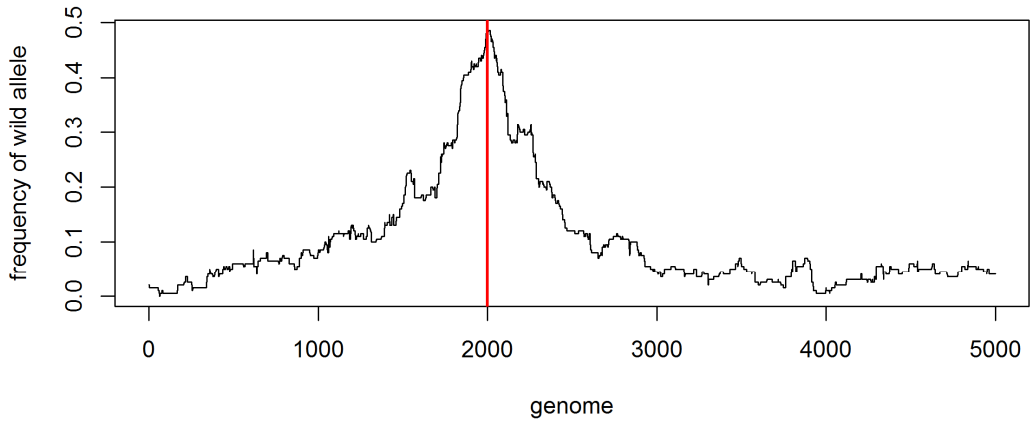
```
# Simulate matings between all founders.
# Each plan is involved in exactly 19 matings.
population <- breeding.diploid(population, breeding.size = c(190,0),
                               breeding.all.combination = TRUE,
                               selection.size = c(20,0),
                               selection.m.cohort = "F0", name.cohort = "F1")

# Simulate matings between plants of the last generation.
# Each plant is involved in exactly 2 matings.

population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = c(2,0),
                               selection.m.cohort = "F1", name.cohort = "F2")
population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = c(2,0),
                               selection.m.cohort = "F2", name.cohort = "F3")
population <- breeding.diploid(population, breeding.size = c(190,0),
                               selection.size = c(190,0), same.sex.activ = TRUE,
                               same.sex.sex = 0, max.offspring = c(2,0),
                               selection.m.cohort = "F3", name.cohort = "F4")
```

16.2 Simulation of Introgression on blue eggshell QTL

We here show how to perform an exemplary simulation of a breeding scheme to perform introgression of a single QTL. In term of computing time this simulation with a 5M genome, 5k SNPs and a total of 520 individuals took 1.2 seconds on one core of my local maschine without the usage of miraculix.

```
# Generate an input SNP-dataset

# 10 White-Layer (0) (20 haplotypes, 5'000 SNPs)
# 10 Wild population (1) (20 haplotypes, 5'000 SNPs)
dataset1 <- matrix(0, nrow = 5000, ncol = 20)
dataset2 <- matrix(1, nrow = 5000, ncol = 20)

# Generation of a trait
# Colums code: SNP, chromosome, effect 00, effect 01, effect 11
# Blue Eggshell QTL is positioned on SNP 2000, chromosome 1
major_qtl <- c(2000, 1, 0, 10000, 20000)
# In all other positions the white layer genome is assumed to be favorable
# All marker effects combiened are smaller than the blue eggshell QTL
rest <- cbind(1:5000, 1, 1, 0.5, 0)
trait <- rbind(major_qtl, rest)

# Generation of the base-population
# First 10 individuals are female (sex=2)
# Next 10 individuals are male (sex=1)
population <- creating.diploid(dataset = cbind(dataset1, dataset2),
                               real.bv.add = trait, name.cohort = "Founders",
                               sex.s = c(rep(2,10), rep(1,10)))

# Simulate random mating:
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(10,10),
                               selection.m.cohorts = "Founders_M",
                               selection.f.cohorts = "Founders_F",
                               name.cohort = "F1")

# Simuation of matings with selection:
# Top 50 cocks are mated to the 10 founder hens
# Selection of the cocks based on their genomic value ("bv")
# Target: Increase share of white layer while preserving blue egg shell QTL
```

```
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(50,10),
                               selection.m.cohorts = "F1_M",
                               selection.f.cohorts = "Founders_F",
                               name.cohort = "BC1", selection.m = "function",
                               selection.criteria.type = "bv")
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(50,10),
                               selection.m.cohorts = "BC1_M",
                               selection.f.cohorts = "Founders_F",
                               name.cohort = "BC2", selection.m = "function",
                               selection.criteria.type = "bv")
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(50,10),
                               selection.m.cohorts = "BC2_M",
                               selection.f.cohorts = "Founders_F",
                               name.cohort = "BC3", selection.m = "function",
                               selection.criteria.type = "bv")

# Mating of cocks and hens that are heterozygous in blue egg shell QTL
# 25% of resulting offspring should be homozygous in blue egg shell QTL

population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(50,50),
                               selection.m.cohorts = "BC3_M",
                               selection.f.cohorts = "BC3_F",
                               name.cohort = "IC", selection.m = "function",
                               selection.criteria.type = "bv")

# Check genomic share of wild race in the final generation
genoIC <- get.geno(population, cohorts = "IC_F")
plot(rowSums(genoIC)/200, xlab = "genome", ylab = "frequency of wild allele", type
= "l")
abline(v = 2000, lwd = 2, col = "red")
```



As expected, the frequency of genetic material stemming from the wild type is higher in the region of the QTL.

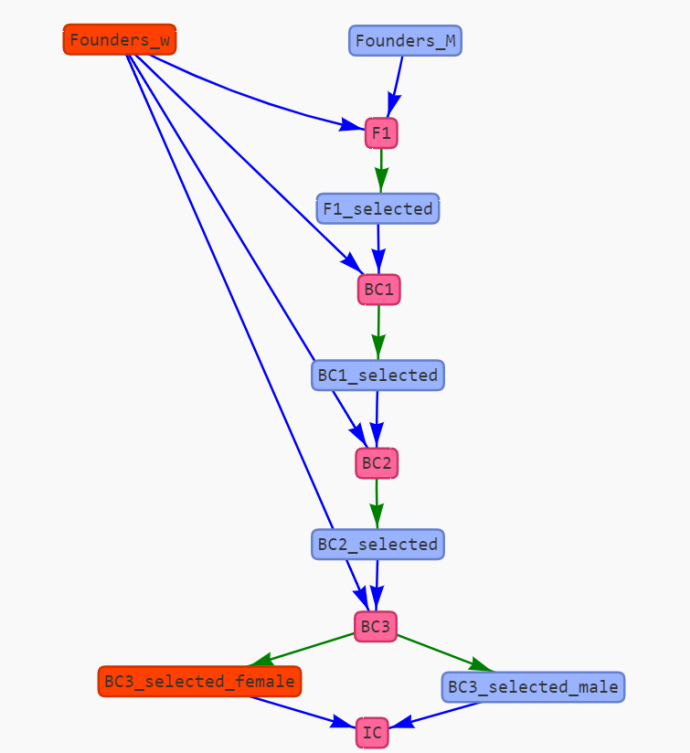


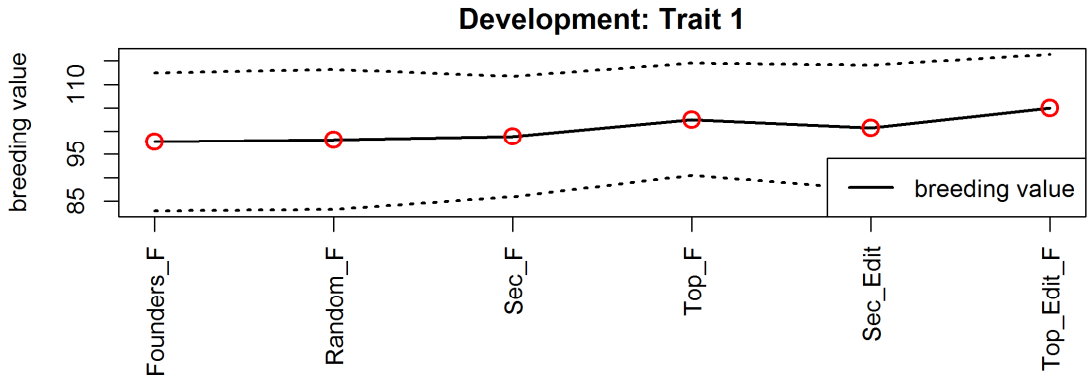
Figure 2: Mating Scheme for Introgression of the blue-egg-shell QTL. Graph is generated via user-interface in MoBPS

16.3 Simulation of gene editing in a cow breeding program

The following script can be used to simulate a breeding program that is utilizing genome editing. Design is chosen according to (Jenko et al. 2015; Simianer et al. 2018). Note that individual numbers are much smaller than in the two references to ensure low computation times. Simulation of 20 generations with 50'000 cows per generation would take 26.3 hours using 24 cores on the gwdg-hpc (Intel E5-2650 (2X12 core 2.2GHz)). This small example with a 5 Morgan chromosome, 5k SNPs and 4300 individuals took 7.4 seconds.

```
# Generation of a base population:
# 1'000 Founder individuals
# 5'000 SNPs
# 100 additive single marker QTL
population <- creating.diploid(nindi = 1000, nsnp = 5000,
                               n.additive = 100, name.cohort = "Founders")

# Simulation of a random mating generation
# 100 bulls (sex=1), 1'000 cows (sex=2) are generated
population <- breeding.diploid(population, breeding.size = c(100,1000),
                               selection.size = c(500,500),
                               selection.m.cohorts = "Founders_M",
                               selection.f.cohorts = "Founders_F",
                               name.cohort = "Random")

# Generate 200 offspring of both from the top 5 bulls / 200 cows
# Heritability of the trait is set to 0.5
# only phenotypes previously unobserved cows are generated
population <- breeding.diploid(population, breeding.size = 200,
                               selection.size = c(5,200), bve = TRUE,
                               heritability = 0.5,
                               new.bv.observation = "non_obs_f",
                               selection.m = "function", name.cohort = "Top",
                               selection.m.cohorts = "Random_M",
                               selection.f.cohorts = "Random_F")

# Generate additional cows using all cows of the previous generation
# Cows are added to the same generation as the previous simulation
population <- breeding.diploid(population, breeding.size = c(0,900),
                               selection.size = c(5,1000),
                               selection.m = "function", name.cohort = "Sec_F",
                               selection.m.cohorts = "Random_M",
                               selection.f.cohorts = "Random_F",
                               add.gen = 3)

# Same cycle as before with additional genome editing
# Edits are chosen based on highest effects in rrBLUP
population <- breeding.diploid(population, breeding.size = c(100,100),
                               selection.size = c(5,200), bve = TRUE,
                               new.bv.observation = "non_obs_f",
                               selection.m = "function",
                               name.cohort = "Top_Edit",
                               selection.m.cohorts = "Top_M",
                               selection.f.cohorts = c("Top_F","Sec_F"),
                               nr.edits = 20, estimate.u = TRUE)

population <- breeding.diploid(population, breeding.size = c(0,900),
                               selection.size = c(5,1000),
                               selection.m = "function", name.cohort = "Sec_Edit",
                               selection.m.cohorts = "Top_M",
                               selection.f.cohorts = c("Top_F","Sec_F"),
                               add.gen = 4)


bv.development(population, cohorts = c("Founders_F", "Random_F", "Sec_F",
                                       "Top_F", "Sec_Edit", "Top_Edit_F"),
```

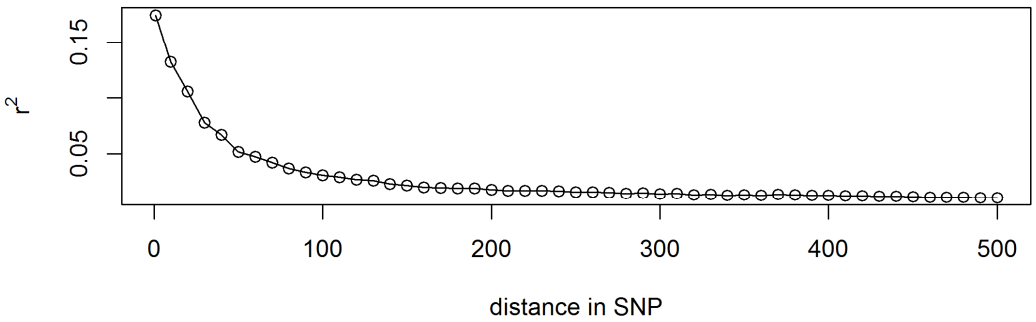
```
display.cohort.name = TRUE, display.sex = TRUE, development = 1)
```



16.4 Simulation of a base population with a hard sweep

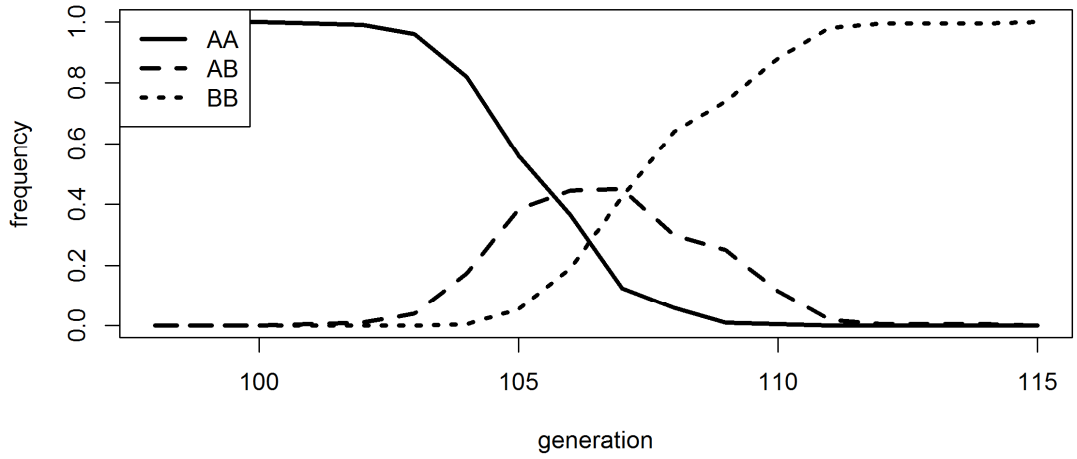
We here show how to perform an exemplary simulation to generate a base population and a hard sweep.

```
# Generate a starting population with 5000 SNPs and 200 individuals
# and a single chromosome of length 2 Morgan.
population <- creating.diploid(nsnp = 5000, nindi = 200, chromosome.length = 2)

# LD build up via 100 generations of random mating
# Each generation contains 200 individuals
for(index in 1:100){
  population <- breeding.diploid(population, breeding.size = 200,
                                 selection.size = c(100,100))
}

# Derive allele frequency and check LD for the last generation:
genotype.check <- get.geno(population, gen = length(population$breeding))
p_i <- rowMeans(genotype.check)/2
ld.decay(population, genotype.dataset = genotype.check, step = 10, max = 500)
```




```
# Simulate a favorable mutation in a previously fixed marker
fixated_markers <- which(p_i==0) # Which markers are fixated
qtl_posi <- sample(fixated_markers, 1) # Selected a fixated marker in A
trait <- cbind(qtl_posi, 1, 0, 1, 2) # SNP, Chromosome, Effect AA, Effect AB,
Effect BB
population <- creating.trait(population, real.bv.add = trait)
```

```
# Generate a mutation in the first male individual
population <- mutation.intro(population, 101, 1, 1, qtl_posi)

# Simulate generations with selection pressure
# Individuals with the favorable SNP are picked 5 times as often
for(index in 1:25){
  population <- breeding.diploid(population, breeding.size = 200,
                                 selection.size = c(100,100),
                                 best.selection.ratio.m = 5,
                                 best.selection.ratio.f = 5)
}

analyze.population(population, gen = 98:115, chromosome = 1, snp = qtl_posi)
```



References

Bekele, Wubishet A.; Wieckhorst, Silke; Friedt, Wolfgang; Snowdon, Rod J. (2013): High-throughput genomics in sorghum. From whole-genome resequencing to a SNP screening array. In: *Plant biotechnology journal* 11 (9), S. 1112–1125.

Browning, Brian L.; Zhou, Ying; Browning, Sharon R. (2018): A One-Penny Imputed Genome from Next-Generation Reference Panels. In: *The American Journal of Human Genetics* 103 (3), S. 338–348.

Groenen, Martien A.; Wahlberg, Per; Foglio, Mario; Cheng, Hans H.; Megens, Hendrik-Jan; Crooijmans, Richard PMA et al. (2009): A high-density SNP-based linkage map of the chicken genome reveals sequence features correlated with recombination rate. In: *Genome Research* 19 (3), S. 510–519.

Jenko, Janez; Gorjanc, Gregor; Cleveland, Matthew A.; Varshney, Rajeev K.; Whitelaw, C. Bruce A.; Woolliams, John A.; Hickey, John M. (2015): Potential of promotion of alleles by genome editing to improve quantitative traits in livestock breeding programs. In: *Genetics Selection Evolution* 47 (1), S. 55.

Lee, Michael; Sharopova, Natalya; Beavis, William D.; Grant, David; Katt, Maria; Blair, Deborah; Hallauer, Arnel (2002): Expanding the genetic map of maize with the intermated B73× Mo17 (IBM) population. In: *Plant Molecular Biology* 48 (5-6), S. 453–461.

Legarra, Andres; Christensen, Ole F.; Aguilar, Ignacio; Misztal, Ignacy (2014): Single Step, a general approach for genomic selection. In: *Livestock Science* 166, S. 54–65.

Liu, Jiajun; Luo, Wei; Qin, Nana; Ding, Puyang; Zhang, Han; Yang, Congcong et al. (2018): A 55 K SNP array-based genetic map and its utilization in QTL mapping for productive tiller number in common wheat. In: *Theoretical and Applied Genetics* 131 (11), S. 2439–2450.

Ma, Li; O'Connell, Jeffrey R.; VanRaden, Paul M.; Shen, Botong; Padhi, Abinash; Sun, Chuanyu et al. (2015): Cattle sex-specific recombination and genetic control from a large pedigree analysis. In: *PLoS genetics* 11 (11), e1005387.

Martini, Johannes W. R.; Gao, Ning; Cardoso, Diercles F.; Wimmer, Valentin; Erbe, Malena; Cantet, Rodolfo J. C.; Simianer, Henner (2017): Genomic prediction with epistasis models: on the marker-coding-dependent performance of the extended GBLUP and properties of the categorical epistasis model (CE). In: *BMC bioinformatics* 18 (1), S. 3.

Meuwissen, T. H. E. (1997): Maximizing the response of selection with a predefined rate of inbreeding. In: *Journal of animal science* 75 (4), S. 934–940.

Microsoft Corporation and Steve Weston (2018): doParallel: Foreach Parallel Adaptor for the 'parallel' Package. Online verfügbar unter https://CRAN.R-project.org/package=doParallel.

Miesenberger, Josef (1997): Zuchtzieldefinition und Indexselektion für die österreichische Rinderzucht: na.

Prieur, Vincent; Clarke, Shannon M.; Brito, Luiz F.; McEwan, John C.; Lee, Michael A.; Brauning, Rudiger et al. (2017): Estimation of linkage disequilibrium and effective population size in New Zealand sheep using three different methods to create genetic maps. In: *BMC Genetics* 18 (1), S. 68.

Purcell, Shaun; Neale, Benjamin; Todd-Brown, Kathe; Thomas, Lori; Ferreira, Manuel A. R.; Bender, David et al. (2007): PLINK. A tool set for whole-genome association and population-based linkage analyses. In: *The American Journal of Human Genetics* 81 (3), S. 559–575.

Renaud Gaujoux (2018): doRNG: Generic Reproducible Parallel Backend for 'foreach' Loops. Online verfügbar unter https://CRAN.R-project.org/package=doRNG.

Rohrer, Gary A.; Alexander, Leeson J.; Keele, John W.; Smith, Tim P.; Beattie, Craig W. (1994): A microsatellite linkage map of the porcine genome. In: *Genetics* 136 (1), S. 231–245.

Simianer, Henner; Pook, Torsten; Schlather, Martin (2018): Turning the PAGE - the potential of genome editing in breeding for complex traits revisited. In: *World Congress on Genetics Applied to Livestock*, S. 190.

VanRaden, Paul M. (2008): Efficient methods to compute genomic predictions. In: *Journal of Dairy Science* 91 (11), S. 4414–4423.

Wen, Weie; He, Zhonghu; Gao, Fengmei; Liu, Jindong; Jin, Hui; Zhai, Shengnan et al. (2017): A high-density consensus map of common wheat integrating four mapping populations scanned by the 90K SNP array. In: *Frontiers in plant science* 8, S. 1389.

Zerbino, Daniel R.; Achuthan, Premanand; Akanni, Wasiu; Amode, M. Ridwan; Barrell, Daniel; Bhai, Jyothish et al. (2017): Ensembl 2018. In: *Nucleic acids research* 46 (D1), D754-D761.

Zheng, Chaozhi; Boer, Martin P.; van Eeuwijk, Fred A. (2015): Reconstruction of genome ancestry blocks in multiparental populations. In: *Genetics* 200 (4), 1073-1087.

17 Acknowledgements

This package was developed in the context of the European Union's Horizon 2020 Research and Innovation Program under grant agreement n°677353 IMAGE

Additional thanks goes to the Research Training Group 1644 "Scaling Problems in Statistics" for financing travelling, BMBF project "MAZE – Accessing the genomic and functional diversity of maize to improve quantitative traits" (Grant ID 031B0195) and all members of the Animal Breeding and Genetics Group at the University of Goettingen for all the helpful advice to people with less genetic background and ideas of things to implement.

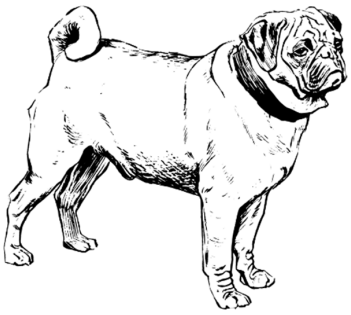


IMAGE
Innovative Management of Animal Genetic Resources

CiBreed
Center for Integrated Breeding Research

GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

C Supplementary figures

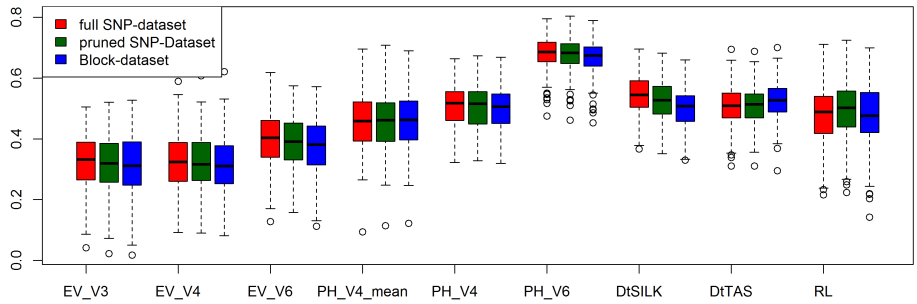


Figure C.1: Accuracy of genomic prediction on the test set using different genomic datasets to derive the genomic relationship matrix (VanRaden, 2008) for Petkuser Ferdinand Rot.

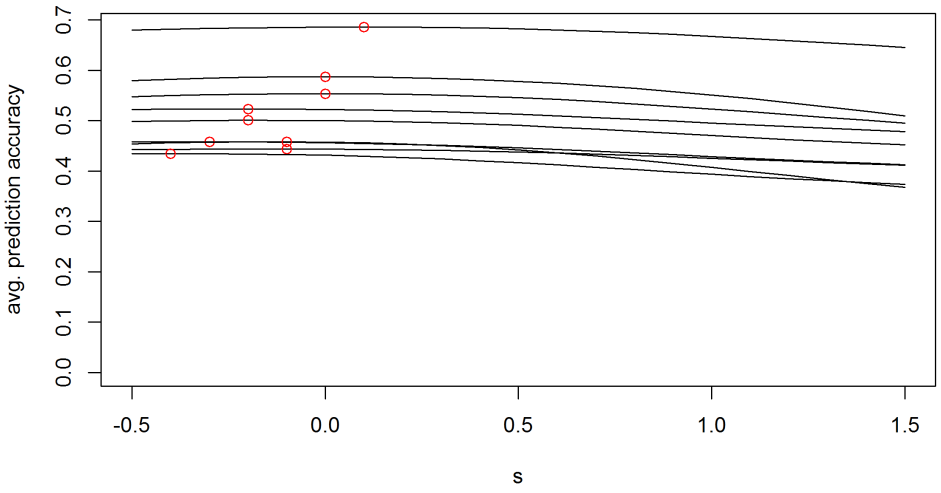


Figure C.2: Accuracy of genomic prediction using different weightings s for the block length when deriving the genomic relationship matrix for Kemater Landmais Gelb. Each line is representing one trait and the red dot is indicating the maximum of the respective curve.

Publications

List of Publications

Pook T, Schlather M, de los Campos G, Mayer M, Schoen CC, Simianer H (2019) HaploBlocker: Creation of subgroup specific haplotype blocks and libraries. Submitted at *Genetics*

Pook T, Schlather M, de los Campos G, Mayer M, Schoen CC, Simianer H (2019) HaploBlocker: Creation of subgroup specific haplotype blocks and libraries. *Biorvix*: doi: https://doi.org/10.1101/339788

Pook T, Mayer M, Geibel J, Weigend S, Cavero D, Schoen CC, Simianer H (2019) Improving imputation quality in BEAGLE for crop and livestock data. Submitted at *G3: Genes, Genome, Genetics*

Pook T, Mayer M, Geibel J, Weigend S, Cavero D, Schoen CC, Simianer H (2019) Improving imputation quality in BEAGLE for crop and livestock data. *Biorvix*: https://doi.org/10.1101/577338

Pook T, Schlather M, Simianer H (2019) MoBPS – Modular Breeding Program Simulator. In preparation.

List of Talks

Pook T (2014) BEAGLE Imputation Algorithm. In: Genetisch Statistischer Ausschuss der DGfZ. Mariaspring, Germany, 01.10.2014

Pook T, Unterseer S, Schoen CC, Simianer H (2017) Creation of haplotype blocks in DH-lines in maize. In: Workshop Biometrische Aspekte der Genomanalyse. Heidelberg, Germany, 04.05.2017

Pook T, Weigend S, Simianer H (2017) A generalized approach to calculate expectation and variance of kinship in complex breeding schemes. In: Annual Meeting of EAAP. Tallinn, Estonia, 30.08.2017

Pook T, Schlather M, de los Campos G, Unterseer S, Schoen CC, Simianer H (2017) Identification of haplotype blocks in DH-lines in maize. In: 4th International Symposium on Genomics of Plant Genetic Resources. Giessen, Germany, 04.09.2017

Pook T, Weigend S, Simianer H (2017) Deterministische Berechnung von Erwartungswert und Varianz des Inzuchtniveaus in komplexen Zuchtprogrammen. In: Vortragstagung der DGfZ und GfT. Hohenheim, Germany, 21.09.2017

Simianer H, Pook T, Schlather M (2018) Turning the PAGE – potential of genome editing in breeding for complex traits revisited. In: World Congress on Genetics Applied to Livestock Production. Auckland, New Zealand, 15.02.2018

Pook T, Schlather M, de los Campos G, Cavero D, Simianer H (2018) Creation of subgroup specific haplotype blocks and libraries. In: World Congress on Genetics Applied to Livestock Production. Auckland, New Zealand, 16.02.2018

Pook T, Schlather M, de los Campos G, Schoen CC, Simianer H (2018) Creation of subgroup specific haplotype blocks and libraries. In: Workshop on Bayesian statistics in population statistics. Hohenheim, Germany, 27.03.2018

Pook T, Schlather M, Simianer H (2018) RekomBre – A tool to simulate and compare large scale breeding programs. In: Annual Meeting of EAAP. Dubrovnik, Croatia, 27.08.2018

Pook T, Mayer M, Simianer H (2018) Analyse der Imputation-,Inferenz- und Phasingqualität in BEAGLE. In: Vortragstagung der DGfZ und GfT. Bonn, Germany, 13.09.2018

Geibel J, Weigend S, Weigend A, Reimer C, Pook T, Simianer H (2018) Auswirkungen des Array Design Prozesses auf die Überschätzung der Heterozygotie. In: Vortragstagung der DGfZ und GfT. Bonn, Germany, 13.09.2018

Pook T, Herzog S, Heise J, Simianer H (2018) Deep Learning – eine Alternative für die Zuchtwertschätzung. In: Genetisch Statistischer Ausschuss der DGfZ. Goettingen, Germany, 19.09.2018

Simianer H, Pook T, Schlather M (2018) MoBPS – ein modularer Ansatz zur Evaluierung komplexer Zuchtprogramme. In: Genetisch Statistischer Ausschuss der DGfZ. Goettingen, Germany, 10.09.2018

Pook T, Herzog S, Heise J, Simianer H (2018) Deep Learning – an alternative for genomic prediction. In: PLANTS and ANIMALS: Bridging the Gap in Breeding Research. Goettingen, Germany, 10.10.2018

List of Posters

Ha NT, Pook T, Dierks C, Weigend S, Preisinger R, Simianer H (2017) A simulation approach to optimize breeding programs with application to the introgression of the blue egg color into a high performing layer line. In: 10th European Symposium on Poultry Genetics. St. Malo, France, 26.-28.6.2017

Pook T, Schlather M, de los Campos G, Schoen CC, Simianer H (2018) Creation of subgroup specific haplotype blocks and libraries. In: PLANT2030 Statusseminar. Potsdam, Germany, 05.-07.02.2018

Geibel J, Weigend S, Weigend A, Reimer C, Pook T, Simianer H (2018) Array design and SNP ascertainment bias. In: World Congress on Genetics Applied to Livestock Production. Auckland, New Zealand, 13.02.2018

Pook T, Schlather M, de los Campos G, Schoen CC, Simianer H (2018) Creation of subgroup specific haplotype blocks and libraries. In: 60th Annual Maize Genetics Conference. St. Malo, France, 22.-25.03.2018

Geibel J, Weigend S, Weigend A, Reimer C, Pook T, Simianer H (2018) Array design and SNP ascertainment bias. In: Population, Evolutionary and Quantitative Genetics Conference. Madison, USA, 13.-16.05.2018

Pook T, Herzog S, Heise J, Simianer H (2019) Deep Learning – an alternative for genomic prediction. In: Gordon Research Conference on “Quantitative Genetics and Genomics”. Lucca, Italy, 10.-15.02.2019

Pook T, Mayer M, Geibel J, Schoen CC, Simianer H (2019) Improving imputation quality in BEAGLE for crop data. In: PLANT2030 Statusseminar. Potsdam, Germany, 13.-15.03.2019

Vogjani E, Martini JWR, Pook T, Schoen CC, Simianer H (2019) Accounting for epistasis improves genomic prediction of phenotypes within and across enviroments. In: PLANT2030 Statusseminar. Potsdam, Germany, 13.-15.03.2019

Curriculum vita

Personal Details

Torsten Pook
An der St.-Vinzenz-Kirche 10 App. 35
37077 Goettingen, Germany
torsten.pook@uni-goettingen.de

Education

2016 – 2019: Ph.D Student in Agricultural Sciences
Georg-August Universität Göttingen, Germany
Thesis: "Methods and software to enhance statistical analysis in large scale problems in breeding and quantitative genetics"

2014 – 2016: Master of Science
University of Mannheim, Germany
Mathematics in Business and Economics
Graduation with distinction
Thesis: "Forecast of daily shipments in logistics"

2011 – 2014: Bachelor of Science
University of Mannheim, Germany
Mathematics in Business and Economics
Thesis: "Analysis of the imputing algorithm BEAGLE"

2002 – 2011: Secondary education (Abitur)
Ratsgymnasium Minden
Germany

Employment

Since July 2016:
Scientific Assistant (Ph.D Student) to Prof. Dr. Henner Simianer
Department of Animal Breeding
University of Goettingen, Germany
Project: MAZE – Accessing the genomic and functional diversity of maize to improve quantitative traits

June 2015 – September 2016:
Working Student
Management board: Finance and Controlling
DB Schenker
Essen, Germany

November 2014 – June 2015:
Research Assistant
Department of Animal Breeding
University of Goettingen, Germany

2012 – 2014:
Research Assistant (Tutor)
Linear Algebra I (HWS12/13 & HWS13/14) and
Calculus II (FSS13)
University of Mannheim, Germany

July 2012 – August 2012
Working Student / Internship
Finance & Controlling
Poppe + Potthoff GmbH

Research stays

October 2017 – December 2017
Institute for Genomic Diversity, Cornell University,
Ithaca, NY, USA,
Prof. Dr. Edward Buckler

Courses

TUM Spring School 2017 "Selection Theory",
Herrsching, Germany,
Prof. Dr. Bruce Walsh

EAAP/CUP Workshop 2017 on
"Writing and Presenting Scientific Papers",
Tallinn, Estonia

Research Interest

Simulation and analysis of breeding programs
Improving genetic imputation
Creation of haplotype block and libraries
Genomic prediction
Artificial neural networks

Awards

Werner-Oettli-Preis 2016
Best Master thesis of the year

EAAP Conference Scholarship 2017
Tallinn, Estonia